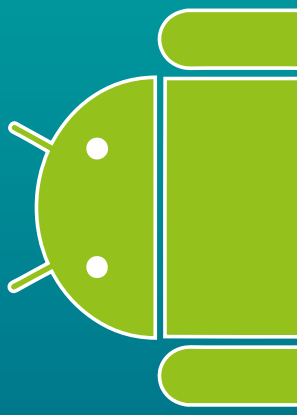
Nr.01 | Frühjahr 2012 | www. ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

ANDROID in der Praxis



JavaOne 2011
Neuigkeiten und Trends

Oracle Public Cloud
Bereit für Wolke sieben

Adobe AIR
Anspruchsvolle Applikationen realisieren

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

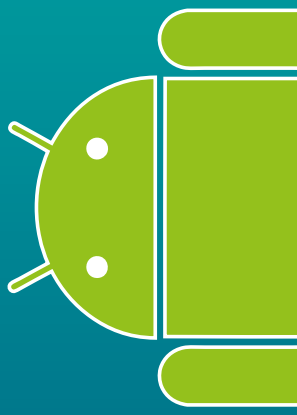
Nr.01 | Frühjahr 2012 | www. ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

ANDROID in der Praxis



JavaOne 2011
Neuigkeiten und Trends

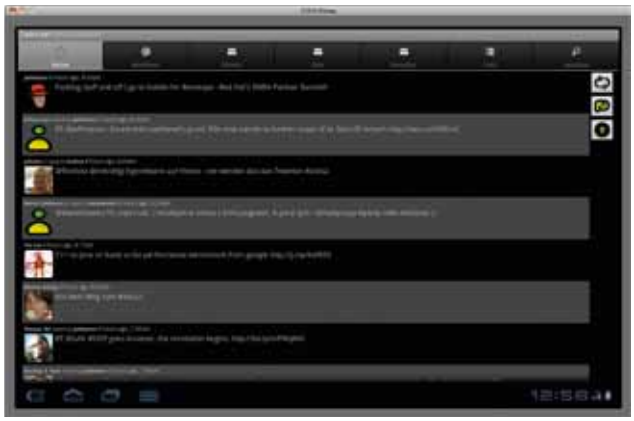
Oracle Public Cloud
Bereit für Wolke sieben

Adobe AIR
Anspruchsvolle Applikationen realisieren

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

Inhalt



Android-Apps fit für die Zukunft machen, Seite 12

Das Treffen der Java-Community auf der JavaOne 2011, Seite 39

Der Weg vom OpenJDK 6 zum OpenJDK 7

Wolfgang Weigend, ORACLE Deutschland B.V. & Co. KG

Das Open Java Development Kit (OpenJDK) basiert auf offenen Standards und einer frei verfügbare Open-Source-Implementierung der Java-Programmiersprache, die auch die offizielle Java Standard Edition (SE) Referenz-Implementierung ist. Die Open-Source-Lizenzierung des OpenJDK erfolgt über GNU General Public License (GPL) v2 für nahezu die gesamte virtuelle Maschine und GPL v2 mit Classpath Exception für die Klassenbibliotheken und Teile der virtuellen Maschine mit den offengelegten Schnittstellen.

Ziel des OpenJDK ist es, eine standardisierte und flexible Software gratis anzubieten, um die IT-Gesamtkosten für die Benutzer niedrig zu halten und maximalen Einsatz offener Standards in Open-Source-Anwendungen oder kommerziellen Software-Umgebungen anzubieten. Die Arbeiten von Oracle am nächsten Release der Java-SE-Plattform – gemeinsam mit der Java Community – sind ein integraler Bestandteil des OpenJDK-Projekts. Für Oracle ist es ein fundamentaler Beitrag zur Open-Source-Java-SE-Implementierung. Mit dem Anspruch, das OpenJDK entscheidend voranzubringen und zur besten Open-Source-Java-Implementierung zu machen, sind externe Entwickler aufgerufen, sich daran zu beteiligen.

OpenJDK im Überblick

Die einheitliche Entwicklungsbasis bildet das offene und frei verfügbare Open JDK als zentrale Grundlage für die Aktivitäten der Java Standard Edition 7 (Java SE 7) und der Java Standard Edition 8 (Java SE 8). Java ist der technologische Ausgangspunkt der meisten Hard- und Software-Hersteller und bildet auch die Basis für kommerzielle Middleware und unternehmensweit genutzte Anwendungen. Dies verdeutlicht auch das Geschäftsmodell für die Java-Entwickler, das anhand der gelernten Programmiersprache und der frei zugänglichen Java-Technologie die von ihnen erstellte Programmierlogik in Form von Anwendungen und neuen Services in die kommerzielle Vermarktung bringt. Die Verwendung von Java in Open-Source-Projekten macht einen Großteil der IT-Landschaft aus, bietet doch der kommerzielle Einsatz des Java-Programmier-Codes die Möglichkeit einer Einnahmequelle für die Entwickler. Bereits bei der Verwendung des OpenJDK ist der Entwickler integraler Bestandteil einer klar umrissenen Java-Strategie. Die neuen OpenJDK-Community-Richtlinien wurden in Zusammenarbeit mit John Duimovich (IBM), Jason Gartner (IBM), Mike Milinkovich (Eclipse), Prof. Doug Lea (State University NY Oswego) und Adam Messinger (Oracle) erstellt und veröffentlicht. Das Gremium hat Regeln aufgestellt, die den langfristigen Bestand und das Wachstum der OpenJDK-Community fördern und sicherstellen, dass die Mitglieder in klarer und offener Weise agieren und die administrative Governance nach dem Leistungsprinzip erfolgt. So wird ein hohes Qualitätsniveau für das OpenJDK erreicht. An dessen Weiterentwicklung sind neben Oracle große Hersteller wie IBM, Apple, SAP, HP sowie Red Hat, Vmware, Azul Systems und Twitter beteiligt. Alle setzen auf die einheitliche Java-Plattform, die aus der Java-Sprache, der Java Virtual Machine (JVM) und den Java-APIs für unterschiedliche Funktionalitäts- und Hardware-Anforderungen wie Java Enterprise Edition (Java EE), Java Standard Edition (Java SE) und Java Micro Edition (Java ME) besteht.

Unter Oracle wurde das JDK-7-Projekt innerhalb des OpenJDK die Referenz-Implementierung für Java SE 7. Das Oracle JDK 7 ist auf der Basis des OpenJDK aufgebaut. Dadurch ist auch der Unterschied zwischen JDK 7 in OpenJDK und Oracle JDK 7 wesentlich geringer geworden. Die überwiegende Mehrheit des JDK-7-Codes ist semantisch gleich, wenn man die beiden Build-Optionen „openjdk = true“ und „openjdk = closed“ (proprietär) miteinander vergleicht. Als generelle Regel gilt: Vom JDK soll so viel wie möglich als Open Source zur Verfügung stehen. JDK 7 umfasst im Wesentlichen Sprachverbesserungen aus dem Projekt Coin, die Concurrency und Collections Updates und die Unterstützung für dynamisch typisierte Sprachen. Die Freigabe vom JDK 8 soll im Jahr 2013 erfolgen und wird die Java-Plattform-Modularisierung mit dem OpenJDK-Projekt „Jigsaw“ und die Lambda-Expressions (Closures) mit dem OpenJDK-Projekt „Lambda“ beinhalten.

Java-Technologie-Komponenten

Der Java Community Process (JCP) ist der inhaltliche und programmatische Rahmen zur Entwicklung von standardisierten technischen Spezifikationen für die Java-Technologie. Interessierte können sich direkt

auf der Java-Community-Process-Webseite registrieren und JCP-Mitglied werden. Sie können bei einer bestimmten JSR-Expert-Gruppe mitmachen, um die einzelnen Java Specification Requests(JSRs) nachzuvollziehen, Feedback zu geben oder eigene JSR-Vorschläge einzureichen. Innerhalb der Regelung des JCP für das Release einer neuen oder überarbeiteten Technologie-Spezifikation müssen drei primäre Komponenten enthalten sein:

a. Spezifikation: Die schriftliche Spezifikation der Technologie liegt vor. Es gibt unterschiedliche Arten von Spezifikationen, beispielsweise:

   – Plattform-Editionen
   – Profile
   – Optionale Packages

b. Referenz-Implementation (RI): Ein Prototyp oder eine Proof-of-Concept-Implementierung der Spezifikation liegt vor. Die Referenz-Implementierung muss vorliegen, um das TCK zu durchlaufen.

c. Technology Compatibility Kit (TCK): Das Test-Kit für Implementierer von Java-Technologie wird verwendet, um sicherzustellen, dass die erbrachte Kodierung mit der Technologie-Spezifikation konform ist. Das TCK muss alle Aspekte der Spezifikation testen, deren Auswirkung betrachten und dabei überprüfen, wie konform die Implementierung der Spezifikation tatsächlich ist. Das betrifft die öffentlichen Schnittstellen (Public APIs) und alle Elemente der Spezifikation.

Die Hersteller-Implementierung einer Spezifikation wird nur dann als konform angesehen, wenn die Implementierung das TCK durchlaufen hat.

Wie Abbildung 1 zeigt, wird für die Entwicklung dieser drei Java-Technologie-Komponenten eine gemeinschaftliche und interaktive Vorgehensweise empfohlen, im Gegensatz zur sequenziellen Herangehensweise.

Ein interaktiver Entwicklungsansatz ist viel effektiver, da für die notwendige Implementierung der Spezifikation die entwickelte Referenz-Implementierung bereits Hinweise zu sinnvollen Änderungen an der Spezifikation liefert. Ein vergleichbarer Vorgang findet statt, wenn das TCK entwickelt wurde und gegen die Referenz-Implementierung getestet wurde. Dann können sich Probleme in der Referenz-Implementierung oder der Spezifikation offenbaren. Der interaktive Entwicklungsprozess, wie mit der Triangel in Abbildung 1 dargestellt, enthüllt Schwachstellen und anfängliche Probleme beim Entwicklungsprozess einer neuen Spezifikation, welche zu einer ganzheitlichen und praxisbezogenen Spezifikation als Ergebnis führen. Daher bietet die interaktive Methode mit gemeinsamer Entwicklung und dem Testen aller drei Komponenten die größtmögliche Effizienz gegenüber der nacheinander ablaufenden, sequenziellen Entwicklungsmethode. Bevor die Entwicklung für das TCK und die Referenz-Implementierung losgeht, wird ein stabiler und belastbarer Entwurf der Spezifikation empfohlen.

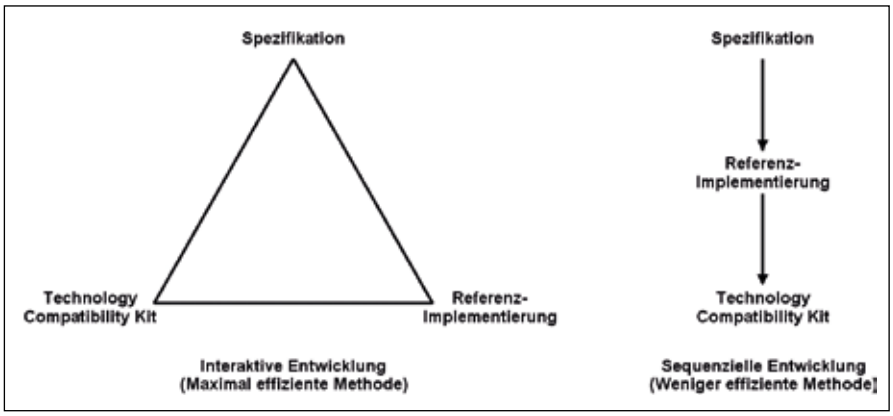


Abbildung 1: Interaktive versus sequenzielle Entwicklung

OpenJDK 6 mit Stammbaum und Entwicklungsvererbung

Die OpenJDK-6-Codebasis ist von der Codebasis für JDK 7 und dem JDK-6-Update-Zweig abgesondert (siehe Abbildung 2). Hier wurden aus der ursprünglichen JDK-6-Codebasis zwei Linien von Erben erzeugt: JDK 7 und die aufeinander folgenden JDK-6-Update-Releases. In einer späten Phase des JDK-6-Entwicklungslebenszyklus fiel die Entscheidung, die JDK-Codebasis als Open Source zur Verfügung zu stellen. Damit wurde ein Build vom JDK 7 erstmals als Open Source veröffentlicht. Nach einiger Zeit wurden das JDK 7 als OpenJDK 7 veröffentlicht und JDK 6u1 als erstes Update nach der Freigabe vom JDK 6 ausgeliefert. Gleichermaßen wurden die Arbeiten am OpenJDK 7 und an den JDK-6-Update-Zweigen fortgesetzt. Zudem wurde eine Open-Source-Implementierung der Java-SE-6-Spezifikation benötigt. Nach Betrachtung der Alternativen wurden OpenJDK 7 Build 20 als Basis für die rückwärtige Erzeugung vom OpenJDK 6 ausgewählt und die für die Java-SE-6-Implementierung ungeeigneten OpenJDK-7-Source-Code-Änderungen entfernt.

Nach diesem Schritt wurden die drei Zweige JDK 6 Update, OpenJDK 7 und OpenJDK 6 kontinuierlich weiterentwickelt. Die einzelnen Versionen wurden mit den notwendigen Fixes versehen oder direkt portiert und alle Release-Zweige enthielten Sicherheits-Fixes. Der OpenJDK 7 Build 147 speiste die Codebasis des OpenJDK 8. Der neue Zweig wächst stetig mit seinen Änderungen und Einträgen.

Aus dem Stammbaum stehen neben den einzelnen JDK Builds auch „OpenJDK 6 Source Bundle Build b23“, „OpenJDK 7 Source Bundle Build b147“ und das sogenannte „JDK 8 Snapshot Release“ als Java SE 8 Developer Preview Releases zum Download von Source-Code bereit.

OpenJDK-Anforderungen

Der OpenJDK-Source-Code wird kontinuierlich verändert und kann jederzeit verwendet werden, um daraus einen „Build“ zu erzeugen. Um diesen Build als „Java“ bezeichnen zu dürfen und Patentschutz dafür zu haben, ist die Technology-Compatibility-Kit-Lizenz (TCK-Lizenz) notwendig, die für OpenJDK-Derivate frei verfügbar

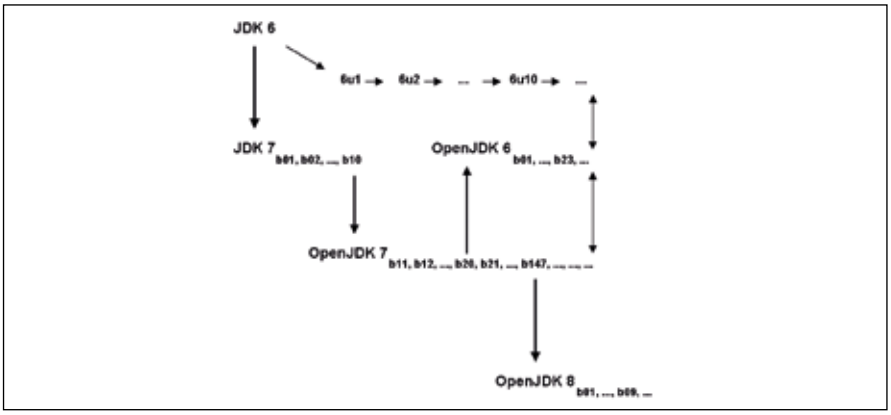


Abbildung 2: Stammbaum und Verzweigungen vom OpenJDK

ist. Dazu muss die Vereinbarung „OpenJDK Community TCK License Agreement Version 1.1" unterzeichnet vorliegen (siehe auch Dokument „openjdk-tck-license.pdf" vom 29. September 2010 oder kurz „TCK Stand-Alone – SE6forOpenJDK"). Die nachfolgende Vereinbarung für das OpenJDK mit Java SE 7 wird unter „OpenJDK Community TCK License Agreement for Java SE 7" auf openjdk.java.net/legal veröffentlicht.

Die eigenständige Wartung, die für ein Hersteller-JDK erbracht werden muss, wenn das OpenJDK als Basis genutzt wird, ist ein großer Kostenblock, der oft nicht bedacht wird. Um beispielsweise ein eigenes Java Release über zehn Jahre auf dem aktuellen technischen Stand zu halten, sind Sicherheits-Patches und Release-Updates einzupflegen. Dabei gibt es keinerlei Unterstützung aus der Community, da es in der Gemeinschaft keine Anforderungen zur Erstellung von Fixes gibt, die wiederum von den Kunden eingefordert werden, die auf der Fehlerbereinigung bestehen. Diese Wartungsarbeiten sind mit einem hohen Aufwand an Ressourcen verbunden:

- Innerhalb von Oracle sind mehrere Hundert Entwickler an Java beteiligt
- Die benötigte Java-Core-Kompetenz am Markt für die Java-Entwicklungsarbeiten ist selten und wird stark nachgefragt
- Organisations- und Prozessstrukturen werden benötigt, um Sicherheitslücken und Zeitzonen-Updates zu erbringen
- Der Einstieg in ein eigenständiges Java-Release wird nur für Großunternehmen empfohlen, sofern diese ein strategisches Investment in Java planen:
  - IBM, SAP, HP, Azul Systems und andere große Hersteller arbeiten mit dem OpenJDK mit sehr großen Java-Entwicklungsteams
  - Google, Twitter und einige wenige Unternehmen mit Spezialanforderungen benutzen das OpenJDK für Cloud-Deployments. Diese Unternehmen sind sehr zurückhaltend, was die Angaben über ihre benötigten Entwicklungsressourcen betrifft, aber diese Kosten werden auf das hohe Geschäftsvolumen umgelegt
  - Twitter ist dem OpenJDK und JCP beigetreten
  - Linux-Distributionen Ubuntu, Fedora und Red Hat Enterprise Linux nutzen OpenJDK als deren reguläre Java-SE-Implementierung
  - Es sind keine weiteren OEM bekannt, die einen Java-Port eigenständig warten und pflegen
- Das OpenJDK wird immer mit der aktuellen Java-Version gewartet und dies entspricht derzeit Java 7. Das bedeutet, bei einem Java-Versions-Upgrade mit Java 8 und Java 9 oder einem Fork sind eigenständige Aufwände zur Pflege notwendig
- Die rechtlichen Anforderungen und Bedingungen der GPLv2-Lizenz müssen eingehalten werden. Neben allgemeinen Lizenzbedingungen schließt dies auch eine Offenlegung des Codes der erstellten Java-Version ein
- TCK Testen und Überprüfen der Kompatibilität ist bei jeder Veröffentlichung einer neuen Java-Version unter Verwendung der „OPENJDK COMMUNITY TCK LICENSE AGREEMENT V 1.1" OCTLA-Bedingungen erforderlich, es ist also obligatorisch

Oracle JDK 7 / Java SE Development Kit 7 Download

Das Java Plattform Standard Edition Development Kit (JDK) steht in der Version 7 von Oracle zum Download bereit. Es ist eine Entwicklungsumgebung zur Erstellung von Anwendungen, Applets und Komponenten unter Verwendung der Java-Programmiersprache und des Java-Programmiermodells. Das JDK enthält Werkzeuge zur Entwicklung und zum Testen von Programmen, die in der Java-Programmiersprache geschrieben und mit der Java-Plattform ablauffähig sind.

Oracle liefert die Portierung für die Microsoft-Windows-Plattform mit 32-bit- und 64-bit-Technologie für unterschiedliche Intel-CPU-Typen seit dem 28. Juli 2011:

- Windows x86 als jdk-7-windows-i586.exe mit 79.48 MB
- Windows x64 als jdk-7-windows-x64.exe mit 80.25 MB

Weitere verfügbare Oracle-Portierungen für JDK 7 sind:

- Linux x86 und Linux x64
- Solaris x86 und Solaris x64
- Solaris SPARC und Solaris SPARC 64-bit

Das Oracle JDK 7 für Mac OS X ist derzeit als Developer Preview Release erhältlich und die globale Freigabe für Oracle JDK 7 auf Apple Mac OS X ist für Sommer 2012 geplant.

OpenJDK-Community-Projekte

Unter den namhaften OpenJDK-Community-Projekten befinden sich OpenJDK 6, JDK 7 (Coin, Da Vinci VM), JDK 8 Features (Lambda, Jigsaw) und die Portierungsprojekte (BSD, IcedTea, Mac OS X, Zero). Das Projekt „IcedTea 2.0" basiert auf dem OpenJDK 7 und bietet das Grundgerüst zur einfachen Erzeugung von OpenJDK-Source-Code (Build) in Verbindung mit frei verfügbaren Build- und Deployment-Werkzeugen. IcedTea enthält alle Änderungen aus dem veröffentlichten OpenJDK7-Zweig mit Sicherheits-Fixes, Bug-Fixes,

Anpassungen für den Zero-Interpreter-Port vom OpenJDK ohne Assembler, mit Anpassungen für den Shark Just-in-Time (JIT) Compiler für Zero und Erweiterungen für die Open-Source-Java-Virtual-Machine JamVM, die mit minimalem Speicherbedarf auf den Betriebssystemen Linux, MAC OS X, BSD-Varianten und Solaris/OpenSolaris zur Verfügung steht.

IcedTea wird derzeit als OpenJDK-Variante mit den GNU/Linux-Distributionen Fedora, Gentoo und Debian ausgeliefert und hat das Ziel, das OpenJDK auf GNU/Linux-Betriebssystemen mit unterschiedlichen Architekturen besser zu unterstützen. Das Projekt konsolidiert die einzelnen Arbeitsbereiche aus den externen Repositories zum vorhandenen OpenJDK-Forest. Das stellt die notwendige Synchronisation mit der OpenJDK-Entwicklung sicher und vereinfacht den organisatorischen Prozess der Rücklieferung von erbrachten Beiträgen zum OpenJDK.

Ausblick

Der etablierte Weg, auf dem Entwickler beim OpenJDK mitmachen können, wird von Oracle fortgeführt. Zur innovativen Verbesserung des OpenJDK sind die interessierten Entwickler angehalten, ihre Ideen mit den aktiven OpenJDK-Entwicklern über die Mailing-Listen zu diskutieren und in die einzelnen Projekte einzubringen. Die aktive Teilnahme am OpenJDK mit den entsprechenden Bedingungen ist unter http://openjdk.org/contribute einzusehen. Absolut notwendig zur Qualitätssteigerung des OpenJDK ist jedoch die zunehmende Teilnahme an Projekten und anfallenden Arbeiten wie die Fehlerbereinigung, Erstellung von Bug-Fixes und Erzeugen weiterer Code-Beiträge, wozu alle Entwickler herzlich eingeladen sind.

Wolfgang Weigend
wolfgang.weigend@oracle.com

Wolfgang Weigend, Systemberater für die Oracle Fusion Middleware bei der Oracle Deutschland B.V. & Co. KG, ist zuständig für Java-Technologie und -Architektur mit strategischem Einsatz bei Großkunden. Er verfügt über langjährige Erfahrung in der Systemberatung und im Bereich objektorientierter Softwareentwicklung mit Java. Davor war er mehr als neun Jahre bei der BEA Systems GmbH für strategische Kunden tätig.

Oracle stellt JavaFX 2.0 vor

Die offizielle Pressemeldung von Oracle

Oracle hat auf der JavaOne 2011 die Verfügbarkeit von JavaFX 2.0 bekannt gegeben. Dabei handelt es sich um eine innovative Java-User-Interface-Plattform für Geschäftsanwendungen und den nächsten Schritt in der Evolution von Java als erste Rich-Client-Plattform.

Oracle hat zudem seine Absicht mitgeteilt, einen Vorschlag vorzulegen, nach dem die JavaFX-Plattform Open-Source als neues Projekt in der OpenJDK Community werden soll. Oracle hat vor, zunächst die JavaFX UI Controls und dazugehörige Libraries beizusteuern; es ist geplant, weitere JavaFX-Komponenten in mehreren Phasen folgen zu lassen.

Rich Client Applications, die JavaFX 2.0 nutzen, sind komplett in Java entwickelt, einer der am meisten genutzten Programmiersprachen mit weltweit über neun Millionen Entwicklern. Indem Entwickler Java für die Server- und Client-Seite ihrer Anwendungen verwenden, reduzieren sie die Komplexität ihrer Geschäftsanwendungen.

JavaFX 2.0 ist eine Web-Komponente, die auf einem Webkit basiert, einer führenden Web Rendering Engine, die es Entwicklern erlaubt, native Java-Funktionen und dynamische Funktionen von Web-Technologien nahtlos zu mischen und anzupassen. Entwickler können mit ihren JavaFX-Anwendungen bestehende Java-Libraries nutzen, auf native Systemfunktionen zugreifen oder sich nahtlos zu Server-basierten Java-Plattformen mit Enterprise Edition (Java EE) Middleware-Anwendungen verbinden. Bestehende Java-Swing-Anwendungen können einfach mit neuen Funktionen wie Rich Graphics API, Media Player und Embedded Web Content aktualisiert werden. Mit JavaFX 2.0 erhalten Entwickler die Flexibilität, Anwendungen mit ihren beliebtesten Tools und Programmiersprachen zu erzeugen.

JavaFX 2.0 führt außerdem FXML ein, eine skriptfähige, XML-basierte Markup-Language, um User Interfaces zu entwickeln. Entwickler, die Web-Technologien oder XML-basierte Markup-Sprachen beherrschen, werden FXML einfach erlernen. Es ist leistungsstark für eine Vielzahl von Anwendungen wie Datenvisualisierung und Formular-basierte Geschäftsanwendungen. JavaFX 2.0 gestattet die Verwendung populärer Skriptsprachen, die von Java Virtual Machine unterstützt werden, dazu gehören: Groovy, JRuby und Scala. Damit kombiniert JavaFX 2.0 die Einfachheit dynamischer Programmiersprachen mit dem leistungsstarken Funktionsumfang der Java-Plattform.

Entwickler können auch ihre beliebtesten Java-Entwicklungswerkzeuge wie NetBeans und Eclipse verwenden, um JavaFX Anwendungen zu programmieren. JavaFX-Anwendungen werden entweder als Desktop-Anwendungen eingesetzt oder sicher in einem Browser über das Java Browser Plugin. Darüber hinaus stellt Oracle neue JavaFX-bezogene Projekte und Programme vor:

- Eine öffentliche Betaversion von Java FX 2.0 für Mac OS X ist ab sofort zum Download verfügbar.
- Es gibt ein beschränktes Early-Access-Programm für JavaFX Scene Builder, ein visuelles Layout-Werkzeug für die JavaFX-Plattform. Es erlaubt UI-Screens zu entwickeln, in dem UI-Komponenten einfach von einer Palette in eine Szene verschoben und dort platziert werden. Scene Builder wird Anfang 2012 als öffentliche Betaversion verfügbar sein.

Weitere Informationen unter:
http://javafx.com

Das Java-Tagebuch

Andreas Badelt, Leiter SIG Java, DOAG Deutsche ORACLE-Anwendergruppe e.V.

Das Java-Tagebuch wurde in Ausgabe 2/2010 gestartet, um einen Überblick über die wichtigsten Geschehnisse rund um Java zu geben – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im dritten Quartal 2011.

7. Juli 2011

Java SE 7 Launch

Der Standard Java SE 7 ist zwar noch nicht offiziell verabschiedet und die Referenz-Implementierung (OpenJDK) noch nicht final, aber Oracle feiert zusammen mit großen Teilen der Java-Community schon einmal den bevorstehenden Launch. Dazu sind per Satelliten-Übertragung Redwood Shores, London und Saõ Paulo zusammengeschaltet. Damit sollen die Java User Groups SouJava und LJC präsentiert und eingebunden werden, zwei neue Mitglieder im JCP Executive Committee. In Redwood Shores kommen diverse Vertreter unter anderem von IBM und HP dazu, um die breite Unterstützung zu demonstrieren. Auch die Teilnehmer des Java-Forums Stuttgart sind zumindest als Zuschauer auch per Satellitenübertragung angebunden – nur eine Gewitterzelle über Stuttgart genau zu Beginn der Veranstaltung ärgert die Zuschauer eine Weile.

Adam Messinger von Oracle führt durch das Programm und lässt prominente Beteiligte am Projekt die vielen Details von Java SE 7 erklären. Die Show ist allerdings fast unamerikanisch schlicht.
[http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html]

14. Juli 2011

JavaFX hat einen kleinen Bruder: GroovyFX

Ein Interview auf fxexperience.com dreht sich um ein Projekt, mit dem JavaFX optimal in Groovy integriert werden soll. Grundsätzlich ist die Integration mit Java FX 2.0 schon möglich, da aus Groovy die pure Java-API genutzt werden kann. Aber die Entwickler wollen die Nutzung über die GroovyFX-API möglichst einfach machen und die Stärken von Groovy (zum Beispiel in Bezug auf „domain specific languages") mit JavaFX zusammenbringen. [http://groovy.codehaus.org/GroovyFX].

18. Juli 2011

Java SE 7 finalisiert

Java SE 7 ist nun offiziell akzeptiert worden. Ebenso wie bei der Abstimmung zur „Public Review"-Phase wird die Gelegenheit wieder für Proteste genutzt. Bis auf Google stimmen aber alle EC-Mitglieder für den JSR-336. Da die Spezifikation nun offiziell ist, steht der Freigabe des bereits vorgestellten Oracle JDK nichts mehr im Weg.

20. Juli 2011

Java-Plattform: Weiterer Zuwachs für die Sprachfamilie

Erst vor einigen Wochen ist Gavin King (Vater von Hibernate) mit dem neuen Projekt „Ceylon" an die Öffentlichkeit getreten. Das Projekt wird seit Längerem von einem Team bei RedHat vorangetrieben. Es soll ein neues SDK plus Programmiersprache schaffen, lauffähig auf der Java-Plattform, aber wohl nicht vollständig kompatibel mit Java-Code.

Jetzt kündigt JetBrains (die Firma hinter IntelliJ Idea, TeamCity etc.) das „Projekt Kotlin" an, eine statisch typisierte, Java-kompatible Sprache.

Die Vielfalt der Sprachen für die Java-Plattform wird langsam unübersichtlich. Aber spannend ist es auf jeden Fall, das Rennen um die Java-Nachfolge zu verfolgen.
[http://in.relation.to/Bloggers/Gavin]
[http://confluence.jetbrains.net/display/Kotlin/FAQ]

28. Juli 2011

Das JDK 7 ist offiziell verfügbar

Zehn Tage nach der Verabschiedung der Spezifikation hat das JDK 7 den Status „general availability" erreicht. Wie Chef-Architekt Mark Reinhold in seinem Blog betont, hat der erste und einzige Release-Kandidat ohne Probleme die Tests bestanden. Dann rechnet er vor, wie lange es gedauert hat: vier Jahre, sieben Monate und siebzehn Tage seit Version 6 (damals noch das Sun JDK). Das Warten auf die Version 8 sollte deutlich kürzer werden – es ist ja schon für 2012 geplant.

Das JDK 7 und der Pentium-Bug

Helle Aufregung nach der Veröffentlichung des JDK 7. Apache-Committer Uwe Schindler schreibt an die Mailingliste des Lucene-Projekts, dass es Probleme mit Lucene und Solr gibt, die auf Hotspot-Compiler-Optimierungen von Schleifen zurückzuführen sind. Diese führen unter seltenen Umständen zum Crash der JVM oder – in der Regel schlimmer – zu Berechnungs-

fehlern, die eventuell unentdeckt bleiben. Durch das Ausschalten von Optimierungsoptionen lassen sich die Fehler zwar vermeiden. Dennoch herrscht Unmut in der Community, weil Oracle diesen und andere Bugs mit niedriger Priorität auf die Liste gesetzt und noch nicht weiterbearbeitet hat. Andererseits haben die betroffenen Apache-Projekte auch erst sehr spät auf die „überraschende" Veröffentlichung des JDK 7 reagiert und ihre Tests darauf losgelassen. Fünf Tage vor Veröffentlichung des JDK hatte Oracle dann von dem Fehler erfahren, sich aber dafür entschieden, das JDK dennoch ohne weitere Verzögerung auszuliefern. Blogger wie Cay Horstmann stellen Vergleiche mit dem Pentium-Bug an. 1994 hatte Intel in einer ähnlichen Situation Pentium-Chips ausgeliefert, obwohl bereits bekannt war, dass sie einen Bug in der Berechnung von Gleitkommazahlen enthielten, der in sehr seltenen Fällen auftrat. Glücklicherweise ist Java „nur" Software, sodass Oracle nicht wie damals Intel 485 Millionen Dollar aufwenden muss, um ein korrigiertes JDK auszuliefern.

1. August 2011

Oracle reagiert schnell auf Community-Kritik

Oracle hat reagiert und den von Apache entdeckten Optimizer-Bug auf „high" hochgestuft, was vermutlich bedeutet, dass ein Fix bereits mit JDK7 Update 1 geliefert wird, das eigentlich nur für Sicherheits-Updates vorgesehen ist.

12. August 2011

Ståhl-Blog: Questions & Answers zu JDK 7

Vielleicht als Reaktion auf die vielen Fragen zum Optimizer-Bug veröffentlicht Henrik Ståhl, Oracles oberster Produktstratege für Java, ein kleines Q&A-Dokument in seinem Blog, das sukzessive erweitert werden soll. Eines der initialen Q&A-Pärchen bezieht sich auf die Probleme mit Apache Lucene, die Antwort bestätigt die Vermutung: Der Fix soll mit Update 1 ausgeliefert werden. [http://blogs.oracle.com/henrik/entry/java_7_questions_answers]

30. August 2011

James Gosling verlässt Google

Java-Vater James Gosling verlässt Google bereits nach wenigen Monaten wieder. In seinem Blog „Nighthacks" verkündet er, dass ihm die Entscheidung sehr schwer gefallen sei. Aber das neue Projekt ist offensichtlich zu interessant. Sein neuer Arbeitgeber heißt Liquid Robotics. Die Firma produziert beziehungsweise betreibt eine Flotte von autonomen, surfbrettartigen Robotern, die auf Langzeitreise über die Weltmeere gleiten und Daten sammeln.

28. September 2011

Neue Version des Java Community Process ist angenommen

Die beiden Executive Committees des Java Community Process (JCP) haben den „final draft" des JSR-348 angenommen, die Überarbeitung des JCP selber. Aus dem Committee für Java SE/EE hat sich lediglich Google enthalten und dies damit begründet, dass der JSR nicht die tatsächlichen Probleme des JCP adressiere. VMware nahm nicht an der Abstimmung teil. Im Committee für Java ME nahmen mehrere Mitglieder nicht an der Abstimmung teil beziehungsweise enthielten sich (Vodafone Group Services) mit der Begründung, die Änderungen seien nicht weitreichend genug. Mitglied Sean Sheedy lehnte den JSR sogar ab und begründete dies detailliert unter anderem damit, dass es weiterhin Probleme mit Lizenz-Änderungen geben könne, die Community kein Eigentum an den Ergebnissen der JSRs habe und dass generell ein bestimmtes Mitglied mehr als andere profitiere.

Ein Blick in die neuen Regeln zum JCP bestätigt, dass Oracle das Heft weiter fest in der Hand hat, was nicht anders zu erwarten war. Bei Lizenzfragen zu JSRs hat beispielsweise die Rechtsabteilung des Konzerns das letzte Wort.

Aber insgesamt kann der JSR 348 als Erfolg gesehen werden, wenn man die ursprünglichen Ziele vor Augen hat: Transparenz, (Community-)Beteiligung, Agilität und Steuerung. Und da es nur ein erster Schritt war, bleibt die Hoffnung auf weitere Verbesserungen. So sehen es auch die meisten EC-Mitglieder – inklusive der beiden vertretenen Java User Groups SouJava und LJC – und haben mit entsprechend großer Mehrheit mit „ja" gestimmt.

2. Oktober 2011

Beginn der JavaOne – große Erwartungen und leichte Enttäuschungen

Die JavaOne hat begonnen. Vom 2. bis 6. Oktober 2011 findet die Java-Konferenz in San Francisco statt – wieder zeitgleich zu Oracles OpenWorld und daher auch nicht wie zu Sun-Zeiten im großen Moscone-Konferenzzentrum, sondern über drei Hotels verteilt. Die Erwartungen sind natürlich groß.

Einen kleinen Dämpfer gibt es aber gleich zu Beginn: Oracles Adam Messinger kündigt in seiner Keynote an, dass sich Java SE 8 um ein halbes Jahr verspäten wird, der GA-Termin („general availability") ist erst für Sommer 2013 geplant. Die gewonnene Zeit soll aber genutzt werden, um möglichst viele Features hineinzupacken.

Ein paar positive Nachrichten gibt es aber auch, beispielsweise eine beeindruckende Präsentation von JavaFX in Form eines Spiels, das sowohl auf einem iPad als auch auf einem Samsung Galaxy mit „einem Linux-basierten Betriebssystem" lief (Wie hieß noch gleich das Betriebssystem, das von diesem Suchmaschinenbetreiber entwickelt wurde?). Auch für die MacOS-Freunde gab es zumindest ein bisschen Fortschritt in Form einer Developer Preview des JDK 7 zu sehen. Noch ist aber viel zu tun, offiziell freigegeben werden soll es erst im zweiten Quartal 2012.

7. Oktober 2011

Die JavaOne 2011 ist erfolgreich zu Ende gegangen

Die JavaOne 2011 ist gestern zu Ende gegangen – und kann wohl als Erfolg gewertet werden. Einige kritische Stimmen bemängeln zwar wieder die gleichzeitige Durchführung mit Oracles OpenWorld. Andere sprechen von „Langeweile" in Form des Ausbleibens von Überraschungen Aber die deutliche Mehrheit der Magazine

und Blogs verteilt Lob, teilweise ist sogar Begeisterung zu spüren. Ein bemerkenswerter Kommentar stammt von Ian Skerret (Director of Marketing der Eclipse Foundation): „I [...] get the feeling Oracle has really energized the JUG leaders and the wider Java community [...] Unlike when Sun was in control, I actually feel like the Eclipse community is welcomed into an Oracle-led Java community." Auch JavaFX-Guru Stephen Chin ist enthusiastisch („Oracle has proved that they are going to continue to move the platform forward and support state-of-the art development on Java client technology") und sogar überrascht: Denn Oracle hat angekündigt, JavaFX sukzessive als Open Source freizugeben, im Rahmen des OpenJDK.

Neben den fälligen Sessions zu Java SE 7 und 8 sowie Ausblicken auf 9 (etwa self-tuning und einiges an Cloud-Unterstützung) war natürlich Java EE 7 ein großes Thema. Zudem stand „Cloud" natürlich noch mehr im Vordergrund. Aber auch Java ME erhielt seinen Anteil am Rampenlicht, insbesondere mit der Ankündigung, die Trennung zwischen Java SE und ME (endlich) zu überwinden. Noch weitaus ehrgeiziger ist das von Adam Messinger in einer Keynote angekündigte Avatar-Projekt: Es soll HTML 5 und Java eng integrieren und darüber hinaus unter anderem die Programmier-Modelle von Java ME, SE und EE weiter vereinheitlichen.

Ein kleiner Wermutstropfen war vielleicht die fehlende Beteiligung von Google. Der anhaltende Rechtsstreit führt offensichtlich zu kindischem Verhalten auf beiden Seiten – Google-Angestellten wurde verboten, auf der JavaOne zu präsentieren, Oracle-Vertreter wiederum vermieden kunstvoll die Erwähnung des Namens „Android" bei Präsentationen auf entsprechenden Tablets, als ob es sich um den Leibhaftigen persönlich handeln würde.

16. Oktober 2011

Java.net-Umfrage: Die wichtigste Neuigkeit von der JavaOne

Parallel zur JavaOne wurde auf java.net eine neue Umfrage gestartet: Worauf bezog sich die wichtigste Neuigkeit, die auf der JavaOne präsentiert wurde. Ganz vorn landete Java 8 mit 34 Prozent, direkt gefolgt von JavaFX mit 24 Prozent und Java EE mit 18 Prozent. Die Umfrage ist nicht als repräsentativ zu bezeichnen (bei 50 Teilnehmern) und es drängt sich die Frage auf, ob hier tatsächliche Neuigkeiten bewertet wurden. Aber als Stimmungsbild darüber, was die Entwickler interessiert, könnte sie taugen. Da ist es schon überraschend, dass das vielfach totgesagte JavaFX doch einiges an Aufmerksamkeit erhält. Java ME und „embedded" rangieren allerdings – wenig überraschend – ganz am Ende.

20. Oktober 2011

Ein flinker Jenkins scheint Hudson abzuhängen

Nutzer (und Committer) des Open-Source CI-Servers Hudson mussten sich im Frühjahr entscheiden, ob sie lieber dem Namen treu bleiben oder dem Kern des Entwicklerteams unter dem neuen Namen Jenkins folgen wollten. Seitdem stellen sich viele natürlich die Frage, ob sie die richtige Wahl getroffen haben. Auf „The Server Side" erscheint nun eine Beitrag von Cameron McKenzie, der Indikatoren zum „Rennen" zwischen Hudson und Jenkins präsentiert – zum Beispiel die Anzahl der Mailinglisten-Abonnenten, der bearbeiteten Tickets, der Commits etc. Bei allen angegebenen Indikatoren hat Jenkins – teilweise deutlich – die Nase vorn. Trotzdem bleibt auch McKenzie vorsichtig, eine Vorhersage zu treffen, da hinter Hudson nun immerhin Oracle und Eclipse stehen und der Zeitraum einfach noch zu kurz war.

Andreas Badelt
Andreas.badelt@doag.org

Andreas Badelt ist Berater und Softwareentwickler / -Architekt bei Infosys Limited. Daneben organisiert er seit 2001 die Special Interest Group (SIG) Development bzw. SIG Java der DOAG.

Impressum

Herausgeber:
Interessenverbund der Java User Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner,
redaktion@ijug.eu

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Katja Borgis
DOAG Dienstleistungen GmbH

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

Mediadaten und Preise:
http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf

Druck:
adame Advertising and Media GmbH Berlin
www.adame.de

Java aktuell
Magazin der Java-Community

Android-Apps fit für die Zukunft machen

Heiko W. Rupp, Red Hat

Im Februar 2011 hat Google die Version 3 „Honeycomb" speziell für Tablets freigegeben, gefolgt von Version 4 „Ice Cream Sandwich" im Oktober. Während die meisten Applikationen, die für frühere Versionen geschrieben wurden, ohne Probleme weiterlaufen, geben sie dem Benutzter auf den Geräten mit „Honeycomb" oder „Ice Cream Sandwich" ein angestaubtes Gefühl, da sie aktuelle Änderungen im User-Interface nicht reflektieren. Dieser Artikel zeigt, wie man mit relativ wenigen Änderungen eine Applikation, die für Android 2.x geschrieben wurde, auf Android 3+ heben kann.

Der Autor hatte unter Android 2.1 mit der Entwicklung seines Twitter-Clients „Zwitscher" begonnen (siehe https://github.com/pilhuhn/ZwitscherA) und war Feuer und Flamme, als das Xoom dann endlich erhältlich war. Man konnte zwar schon im Emulator testen, aber zum einen war dieser bei der hohen Auflösung eher gemächlich und zum anderen hat man einfach auch kein echtes Gefühl für das Verhalten der Anwendung. So erfolgt beispielsweise das Drücken eines Buttons im Emulator mit dem Mauszeiger, den man genau positionieren kann, während man auf echten Geräten die Icons nicht zu klein machen darf. Hier wurde dann schnell klar, dass eine Überarbeitung der Anwendung notwendig ist, während sie auf dem Telefon mit seinem beschränkten Bildschirmplatz soweit ganz gut aussieht (siehe Abbildungen 1 und 2).

Eine der deutlich sichtbarsten Änderungen in „Honeycomb" ist die ActionBar, die am oberen Rand des Bildschirms einen Platz für Namen und Icon der Applikation, einen möglichen Untertitel und eine Taste für den Aufruf des Menüs besitzt. Ist in der Leiste Platz, so kann Android auch Menü-Aktionen direkt in der ActionBar zugänglich machen.

Eine weitere deutliche Änderung, die sich aus dem obigen ergibt, ist, dass Anwendungen, die noch nicht auf Android 3 portiert wurden, auf Geräten ohne Hardware-Tasten (Tablets wie das Motorola-Xoom) am unteren Rand eine zusätzliche Soft-Taste für das Menü haben (siehe Abbildung 3).

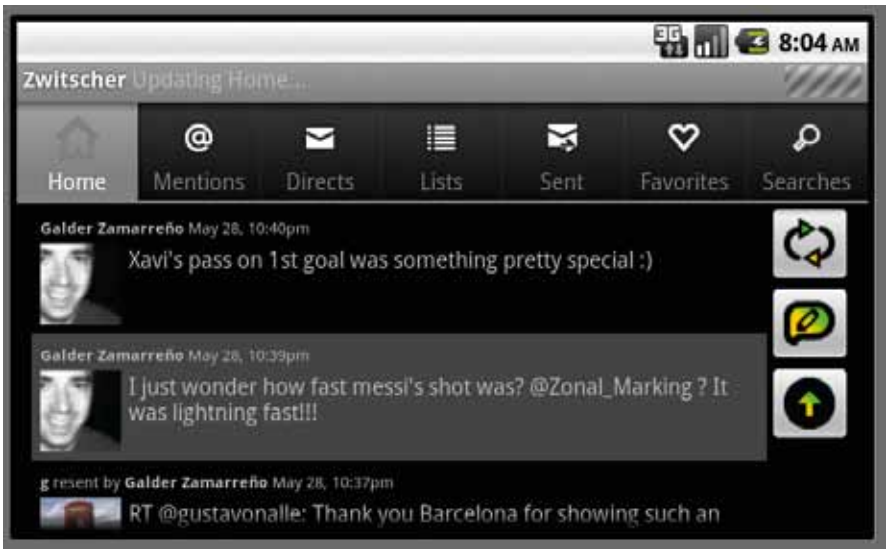


Abbildung 1: Die Anwendung für Android 2 auf dem Mobiltelefon

API-Abfrage

Um die Features von Android 3 oder später nutzen zu können, ist es zunächst notwendig, die SDK-Version im Attribut „targetSdkVersion" in der Datei „AndroidManifest.xml" auf „11" (Honeycomb 3.0) oder größer zu setzen (siehe Listing 1). Die verfügbaren SDK-Versionen stehen in der Tabelle 1 „Android-SDK-Versionen".

```
<manifest
        package="de.bsd.zwitscher">
 ...
    <uses-sdk android:minSdkVersion="8"
            android:targetSdkVersion="11"/>
</manifest>
```

Listing 1

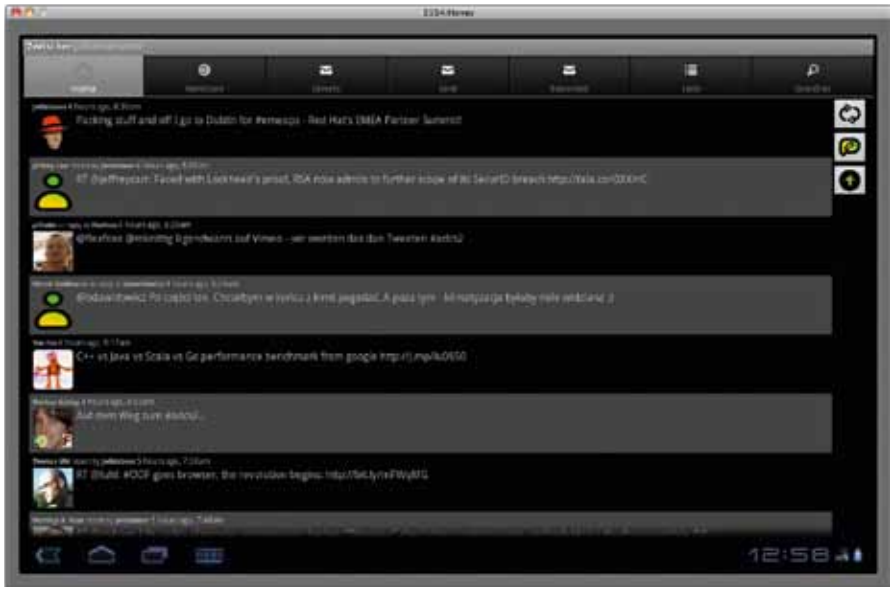


Abbildung 2: Die unveränderte Anwendung auf dem Tablet

Abbildung 3: Die Softkeys zeigen einen extra Knopf für das Menü, wenn die Anwendung nicht auf API-Level 11 oder höher läuft

Abbildung 4: Ausschnitt aus der Menü-Leiste mit Zurück-Indikator, Icon, Titel und Untertitel

Abbildung 5: Ausschnitt der ActionBar mit direkt zugänglichen Menü-Einträgen und dem Menü-Indikator auf „Honeycomb“

Die Änderung der Target-SDK-Version lässt nicht nur den zusätzlichen Taster für das Menü verschwinden, sondern schaltet auch das Handling der neuen Position des Menüs in der ActionBar frei - dazu nachfolgend mehr. Innerhalb des Codes kann die laufende SDK-Version über die Konstante „android.os.Build.VERSION.SDK_INT“ ermittelt werden, sodass Code ab Version 11 die neuen Features nutzen kann, während man für frühere Versionen den alten Code beibehält (siehe Listing 2).

```
if (android.os.Build.VERSION.SDK_INT< 11) {
   // alter Code für vor HC
} else {
  // Code für HC
}
```

Listing 2

Auf den älteren Geräten (ab der im Manifest angegebenen minSdkVersion) laden die Anwendungen wie gewohnt; der Code für „Honeycomb“ wird einfach nicht ausgeführt und gerät damit auch nicht in Probleme, wenn die entsprechenden Methoden nicht vorhanden sind. Mittlerweile ist es möglich, im Markt verschiedene Binarys für unterschiedliche SDK-Versionen einer Applikation einzustellen, sodass man nicht notwendigerweise dies alles in eine Anwendung einkodieren muss; dies war jedoch beim Start von Android 3 nicht möglich. Ein Split der Binary ist hier klarer im Kodieren, hat aber auch den Nachteil, dass man immer mehrere Versionen pflegen muss. Die beschriebene Code-Weiche kann auch genutzt werden, wenn man unterschiedliche Layout-Ressourcen für eine Aktivität je nach API-Level laden möchte.

Auswahl

Nun, da wir die alte Menü-Taste haben verschwinden lassen, ist es Zeit, ein Menü speziell für „Honeycomb“ und später anzulegen. Im Code müssen wir wieder die Android-Version checken, ein Menü aus der XML-Definition (siehe Listing 3) erzeugen und dieses der ActionBar hinzufügen.

Über „actionBar.setDisplayHomeAsUp Enabled(true)“ teilen wir dem System mit, dass der Benutzer durch Druck auf das Home-Icon auf den Schirm davor gelangen kann. Dies wird dann mit einem kleinen Winkel neben dem Programm-Icon angezeigt (Variante ab 3.1; in 3.0 gab es noch ein kleines Dreieck links oben in der Ecke, siehe Abbildung 4).

Abgefragt wird diese „Zurück-Taste“ über den Callback für Menüs:

```
public boolean
onOptionsItemSelected(MenuItem item)
{
 switch (item.getItemId()) {
    case android.R.id.home:
        ....
```

Die im obigen Code-Schnipsel angegebene Menü-Ressource erhält ein neues Attribut, das festlegt, ob der Menü-Eintrag in der ActionBar erscheinen soll oder nicht:

```
android:showAsAction="withText"
```

Mögliche Werte sind dabei:

- withText: es soll nicht nur das Icon angezeigt werden, sondern auch der Alternativtext
- ifRoom: Anzeige nur, falls Platz in der ActionBar ist
- always: immer in der ActionBar anzeigen
- never: soll nie in der ActionBar angezeigt werden

Die einzelnen Werte können dabei mithilfe des Pipe-Symbols kombiniert werden (z.B. ifRoom|withText). Innerhalb der ActionBar werden die Menü-Einträge von links nach

```
public boolean onCreateOptionsMenu(Menu menu) {
  if (android.os.Build.VERSION.SDK_INT>=11) {
          MenuInflater inflater = getMenuInflater();
          inflater.inflate(R.menu.user_detail_menu_honey,menu);

          ActionBar actionBar = this.getActionBar();
          actionBar.setDisplayHomeAsUpEnabled(true);
     return true;
  }
    return false;
}
```

Listing 3

```
<ProgressBar xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/ProgressThingy"
    />
```

Listing 4

```
public MyActivity extends Activity {
   ProgressBar pg;

   public boolean onCreateOptionsMenu(Menu menu) {
       if ( Build.VERSION.SDK_INT>=11)) {
          MenuInflater inflater = getMenuInflater();
          inflater.inflate(R.menu.demo,menu);
          pg = (ProgressBar) menu.findItem(R.id.ProgressBar).getActionView();
     }
  }
```

Listing 5

```
    <style name="MyActionBar" parent="android:style/Widget.
Holo.ActionBar">
        <item name="android:background">#446644</item>
    </style>
```

Listing 6

```
    <style name="ZwitscherStyle" parent="android:style/Theme.Holo">
        <item name="android:actionBarStyle">@style/MyActionBar</item>
    </style>
```

Listing 7

rechts in der Reihenfolge, in der sie in der XML-Menü-Definition aufgelistet sind. Abbildung 5 zeigt einen Teil der ActionBar mit zwei direkt zugänglichen Menü-Einträgen und dem Indikator für das Menü.

Eigenen Titel generieren

Viele Anwendungen nutzen mit Android 2 die Titelleiste zur Anzeige von Fortschrittsbalken oder Detail-Informationen zur aktuellen Activity und legen dazu eine „Custom Title Bar" an. Sobald API 11 oder höher angewählt wurde, funktioniert dies nicht mehr und die Applikation crasht, da es in dem Sinne keine Titlebar mehr gibt, sondern eben die ActionBar. Hier muss man wieder die API-Weiche benutzen, um die Custom Title Bar nur für API-Level kleiner als 11 einzusetzen. Für Texte, die angezeigt werden sollen, ist ein Ersatz auch einfach via „ActionBar.setSubtitle(string)" erhältlich. Schwieriger wird es für den Fortschrittsbalken, der so erst mal nicht mehr in der ActionBar existiert. Dies lässt sich allerdings durch einen speziellen Menü-Eintrag lösen:

```
<menu >
 <item android:id="@+id/ProgressBar"
  android:actionLayout="@layout/pg"
  android:visible="true"
  android:showAsAction="always"
 />
```

Listing 4 zeigt die dazugehörige Layout-Ressource. Dies gibt dann statt eines Fortschrittbalkens einen Kreis. Lustig wird es, wenn man die Höhe und Breite unterschiedlich angibt.

Um die ProgressBar sichtbar zu machen, muss man ein wenig Klimmzüge machen und eine Variable anlegen, in die dann beim Anlegen des Menüs als Referenz auf die ProgressBar abgelegt wird (siehe Listing 5).

ActionBar mit Stil

Im gezeigten Fall entsprach die Standard-ActionBar nicht dem Geschmack des Autors und die existierenden, teilweise transparenten Icons sahen auch nicht gut aus, sodass ein wenig Styling notwendig war. Der erste Schritt besteht darin, einen eigenen Stil für die ActionBar festzulegen (in der Datei „res/values/style.xml"), der auf

der Standard-Definition aufbaut (siehe Listing 6).

Dieser Stil kann dann im nächsten Schritt als Teil eines eigenen Stils verwendet werden (siehe Listing 7).

Er kann dann in AndroidManifest.xml als Stil für die Applikation eingebunden werden (siehe Listing 8).

Netzwerken

In Android-Versionen vor „Gingerbread" konnte man bezüglich lang dauernder Aktionen im Prinzip tun und lassen, was man wollte, im Zweifelsfall auch bei den Netzwerk-Zugriffen. Im schlimmsten Fall bekam der Benutzer in einem Dialog zu sehen, dass die Anwendung nicht antwortet, auch als „Application Not Responding" (ANR) bekannt. Android 2.3 führte dann den StrictMode ein, um Aktionen bei solchen potenziell langsamen Operationen auszuführen. Dieser ist in erster Linie eine API, die dem Entwickler helfen soll, langsamen Code zu finden. „Honeycomb" geht hier einen Schritt weiter: Sobald die Anwendung versucht, im UI-Thread Netzwerk-Aufrufe auszuführen, wirft das System eine „NetworkOnMainThreadException". Glücklicherweise lassen sich diese Netzwerkaufrufe via „AsyncTasks" in den Hintergrund verlagern (siehe Listing 9).

Die wesentliche (und einzige benötigte) Methode dabei ist „doInBackground()", die automatisch in einem eigenen Thread ausgeführt wird. Dies ist also der Platz für lange laufende Aktionen und Netzwerk-Zugriffe. Bevor das System diese Methode aufruft, wird im UI-Thread der „onPreExecute()"-Callback aufgerufen, in dem man beispielsweise einen „Bitte-Warten"-Dialog öffnen oder – im Fall von „Honeycomb" und später – den Untertitel in der ActionBar setzen kann. Das Gegenstück ist „onPostExecute()", das nach „doInBackground()" aufgerufen wird und wieder im UI-Thread läuft, um dann den Dialog zu schließen und auch die Ergebnisse des Netzwerk-Zugriffs in der Benutzeroberfläche anzuzeigen.

Fazit

Mit den beschriebenen Änderungen läuft eine Anwendung nativ sowohl auf „Froyo" als auch auf „Honeycomb" und nutzt die jeweiligen Features aus (siehe Abbildung 6).

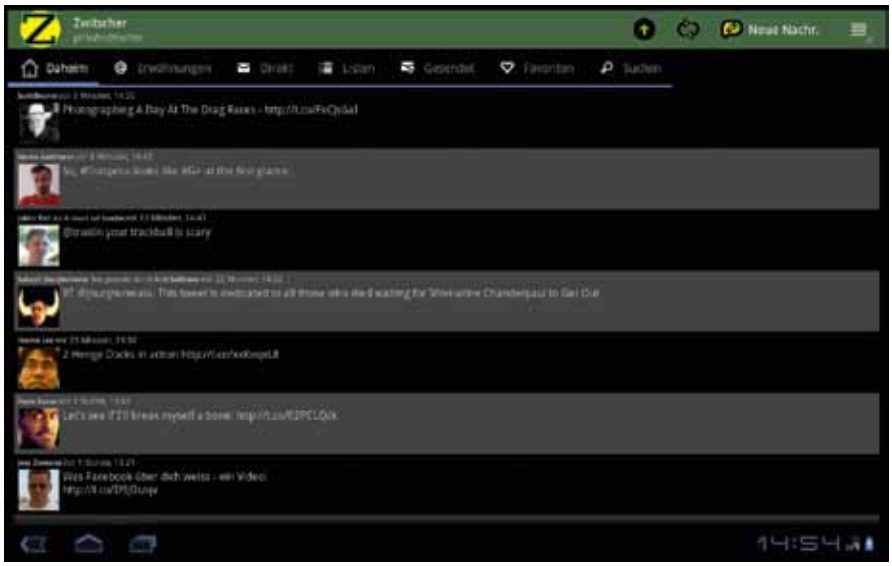


Abbildung 6: Die Anwendung mit den oben beschriebenen Änderungen auf Honeycomb 3.0

```
<manifest ... >
    <application android:icon="@drawable/icon"
                 android:label="@string/app_name"
                 android:theme="@style/ZwitscherStyle" >
```

Listing 8

```
public class MyTask extends AsyncTask<Void,Void,String> {
 Activity context;
 public MyTask(Context context) {
   this.context=context;
 }

 protected String doInBackground(Void... voids) {
    // Netzwerkzugriff
   String x = ....
   return x;
 }

 protected void onPreExecute() {
   context.getActionBar().setSubTitle(„Updating");
 }

 protected void onPostExecute(String message) {
   context.getActionBar().setSubTitle(message);
 }
```

Listing 9

Natürlich hört an dieser Stelle die Arbeit nicht auf. Wie man in Abbildung 6 sieht, wird der Platz auf dem Bildschirm durch diese Änderungen noch nicht optimal ausgenutzt. Hier kann man beispielsweise über Fragmente die Details für einen Tweet, die bei einem Telefon auf einer eigenen Seite gezeigt werden, rechts neben der Liste der Tweets anzeigen. Oder es könnten mehrere Timelines nebeneinander angezeigt werden. Auch die Icons sollte man idealerweise an die neue Bild-

schirmauflösung anpassen, um das Aussehen der Anwendung weiter zu optimieren. Der Zwitscher-Quellcode ist unter https://github.com/pilhuhn/ZwitscherA verfügbar, sodass sich die genannten Beispiele dort nachschauen lassen. Code-Beiträge sind ebenfalls willkommen.

Heiko W. Rupp
heiko.rupp@redhat.com

Heiko Rupp arbeitet bei Red Hat als Entwickler im Bereich System-Management und Monitoring und hier speziell am Jboss-Operations-Network und RHQ-Project.org. Rupp arbeitet und lebt mit seiner Familie in Stuttgart. Seine Hobbys beinhalten Klettern, Lesen und „Vor-dem-Computer-Hocken".

Version	Name	SDK
1.0	Base	1
1.1	Base	2
1.5	Cupcake	3
1.6	Donut	4
2.0	Eclair	5
2.0.1	Eclair	6
2.1	Eclair	7
2.2	Frozen Yoghurt („Froyo")	8
2.3	Gingerbread	9
2.3.3	Gingerbread	10
3.0	Honeycomb	11
3.1	Honeycomb	12
3.2	Honeycomb	13
4.0	Ice Cream Sandwich	14

Tabelle 1: Android-SDK-Versionen

Enterprise JavaBeans 3.1

gelesen von Jürgen Thierack

Mit ihrer geballter Sachkompetenz aus jahrelanger Arbeit mit Java-Komponenten muten die Autoren dem Leser sehr viel zu. Das Buch wendet sich an Leser, die wirklich alles von Grund auf wissen wollen, was es mit jenen standardisierten Komponenten („Beans") auf sich hat, in denen Unternehmen (Enterprises) ihre Geschäftslogik auf Web-Servern codieren. Die Autoren mühen sich um Auflockerung, indem sie öfter mal in den Plauderstil verfallen: „Wenn wir jeden Kaffeeküchen-Besuch mit den gleichen Worten beenden wollten, dann würden die wohl lauten: ‚Im Übrigen sind wir der Meinung, dass jede EJB-Anwendung Unit-getestet werden muss.'" (Seite 348).

Der Untertitel nennt Ein- und Umsteiger. Was ein Einsteiger ist, können wir uns sofort vorstellen. Aber von woher kommt der „Umsteiger"? Er kommt von Version 2 der EJB, könnte also auch „Aufsteiger" heißen. Ein solcher Umsteiger kann Kapitel 1 „Technische Grundlagen und historische Entwicklungen" überspringen und sich stattdessen am Anfang des zweiten Kapitels einen zusammenfassenden Überblick zu den Neuerungen in den Versionen 3.0 und 3.1 (samt Version 2.0 des JPAs) verschaffen.

Wie es sich für ein „Praxisbuch" gehört, werden viele Beispiele (Quellcode per Download) durchgesprochen. Als Plattform dient der kostenlose Open Source Jboss-Application-Server in der Version 6. Die Beispiele kann man manuell per Java-Konsole zum Laufen bringen oder das Build-Tool ANT verwenden. Mit Fragen einer Entwicklungsumgebung beschäftigt sich das Buch nicht, an nur einer Stelle findet die IDE-Eclipse eine kurze Erwähnung. Der Einsteiger kann mit einer EJB-Version von „Hello World" beginnen. Das komplexeste Beispiel ist besonders nett und originell: Das offizielle Begleitspiel „EinStein würfelt nicht!" der Wanderausstellung zum Einstein-Jahr 2005 ist als EJB realisiert. Sehr viele Aspekte der EJB-Entwicklung werden abgedeckt, so kommen alle Bean-Typen zum Einsatz.

Nachdem in Kapitel 11 das Gelernte praktisch erprobt wurde, folgt noch ein abschließendes Kapitel 12, in dem wiederum die Praxis ganz im Vordergrund steht und mit „Kochrezepte" betitelt ist.

Titel:	Enterprise JavaBeans 3.1
Autoren:	W. Eberling, J. Leßner
Verlag:	Hanser München
Umfang:	364 Seiten
Preis:	39,90 €, eBook (downloadbar) inklusive
ISBN:	978-3-446-42259-9

Jürgen Thierack ist in der Software-Branche freiberuflich unterwegs. Seit 10 Jahren liegt sein Schwerpunkt bei Fragen der Finanzmathematik, darunter der Preisfindung für Optionen und Optionsscheine. Aktuelle Thematik: Trendfolgesysteme.

Der Rechtsstreit um Android

Andreas Badelt, Leiter SIG Java, DOAG Deutsche ORACLE-Anwendergruppe e.V.

Im Lauf der letzten Monate hat sich viel getan im Rechtsstreit zwischen Oracle und Google um das Betriebssystem Android – zeitweise gab es täglich Neuigkeiten. Daher haben wir dieses Thema aus dem Java-Tagebuch herausgetrennt, um es hier in gebündelter Form zu präsentieren.

Anfang Juli 2011 verliert Google eine Auktion um Nortel-Patente gegen ein Bieter-Konsortium aus Microsoft, Apple, RIM und anderen. Es wäre eine gute Gelegenheit gewesen, die von allen Seiten – nicht nur von Oracle – mit Patentklagen bedrohte Android-Plattform mit einem größeren, eigenen Patent-Portfolio zu schützen. Etwas später erwirbt Google dann ein größeres Patent-Paket von IBM – die Motivation für „Big Blue" bleibt unklar (ging es nur um ein schnelles Geschäft oder steckt mehr dahinter?). Auch das Potenzial, das sich daraus für Google ergibt, ist für Beobachter schwer einzuschätzen.

Daneben schreiten im Juli 2011die gerichtlichen Vorverhandlungen voran. Google hat beim United States Patent and Trademark Office (USPTO) die Überprüfung der sieben Patente verlangt, die Bestandteil der Klage sind. In vier Fällen hat das USPTO tatsächlich das Patent insgesamt oder einen Teil der Forderungen, die Oracle daraus konstruiert hat, für ungültig erklärt. Dies ist allerdings keine endgültige Entscheidung, sondern kann wiederum von der Gegen-Partei angefochten werden (was in der Vergangenheit häufig erfolgreich war). Gleichzeitig hat Richter Alsup den Vorschlag gemacht, die Streitpunkte aus Effizienzgründen auf den wesentlichen Kern zu reduzieren – was Google natürlich entgegenkommt.

So könnte weiterhin alles auf einen Showdown vor Gericht ab dem 31. Oktober 2011hinauslaufen.

Aber jetzt hat Google wohl zum ersten Mal die Bereitschaft zu einer außergerichtlichen Einigung gezeigt: Bezugnehmend auf die Reduzierung der Streitpunkte schreiben die Google-Vertreter, dass dies auch eine „informelle Lösung" der Angelegenheit erleichtern würde. Wenn es allerdings konkrete Aktivitäten in dieser Richtung geben sollte, dringt zumindest nichts davon an die Öffentlichkeit.

Wenige Tage später entscheidet Richterin Ryu auf Oracles Antrag, dass Googles CEO Larry Page für eine zweistündige Befragung vorgeladen werden kann. Die Befragung soll der Klärung dienen, inwieweit Google Absicht bei der Verletzung von Patenten vorgeworfen werden kann und wie hoch der Wert von Adroid für Google ist. Google hat im Gegenzug den Antrag gestellt, Oracles CEO Larry Ellison vorzuladen.

Ein weiteres Thema ist die Benennung von Experten, die das Gericht beraten sollen. Der Richter hat die Parteien aufgerufen, sich entweder auf eine gemeinsame Liste zu einigen oder jeweils eigene Listen abzugeben. Interessanterweise möchte Google lieber einen Marketing- als einen Technik-Experten benennen, um das Thema „geistiges Eigentum" (intellectual property) in Bezug auf Java zu vertreten. Es wird spekuliert, dass ein Technik-Experte nach Googles Ansicht vermutlich eher die Bedeutung bestimmter Teile von Java für die Android-Plattform betonen würde – was für die Position des Beklagten sicher nicht gut wäre.

Anfang August 2011 versucht Google, den Ausschluss von angeblichen Urheberrechtsverletzungen zu erreichen, bevor der eigentliche Prozess startet, scheitert jedoch einige Wochen später damit. Was noch schwerer wiegen könnte, ist aber der Streit um das „Lindholm-Dokument". In dieser Mail aus dem August 2010, die in verschiedenen Entwurfs-Stadien und als versendete E-Mail gesichert wurde, schreibt Google-Ingenieur Lindholm, dass sie von den Firmen-Gründern Page und Brin gebeten worden seien, technische Alternativen zu Java für Android und Chrome zu untersuchen – dass sie aber zu dem Schluss gekommen seien, dass es keine Alternativen gibt und lieber über eine Lizenz für Java verhandelt werden sollte (diese Lizenz hat Google aber nie erworben, wie bereits bekannt geworden ist).

Die E-Mail hat enorme Sprengkraft, insbesondere wenn es um die Bewertung der Absichtlichkeit von Patentverletzungen geht. Google versucht nun, sie zurückzuhalten: Sie sei, in Vorbereitung auf den Rechtsstreit mit Oracle, unter anderem an einen Vertreter von Googles Rechtsabteilung gegangen und damit nicht als Beweismittel im Prozess zuzulassen. Wie Florian Müller in seinem FOSSPatents-Blog berichtet, gibt es aber auch noch eine um fast fünf Jahre ältere Mail von Android-Gründer Andy Rubin, die bereits eine ähnliche Aussage enthält: „If Sun doesn't want to work with us, we have two options: 1) Abandon our work and adopt MSFT CLR VM and C# language – or – 2) Do Java anyway and defend our decision, perhaps making enemies along the way". Im weiteren Verlauf werden vom Gericht mehr E-Mails öffentlich gemacht, die zum Beispiel von

Verhandlungen mit Sun über ein gemeinsames Open-Source-Geschäftsmodell für Android und Java berichten.

Als Nächstes wird Motorola in den Prozess hineingezogen und soll über Details der Nutzung von Android-Code und Tools aussagen.

Nur wenige Tage später verkündet Google, dass es für 12,5 Milliarden Dollar Motorola Mobility erwerben und als eigenständiges Geschäft weiterführen will. Google-Chef Larry Page verkündet in seinem Blog-Eintrag ganz offen, dass es darum geht, „das Patent-Portfolio von Google zu stärken" und damit die Android-Plattform besser vor „Microsoft, Apple und anderen Firmen" zu schützen. Kommentatoren werfen allerdings auch die Frage auf, was es für die Zukunft von Android insgesamt bedeutet, wenn Google nun mit einem eigenen Unternehmen den anderen Android-Lizenznehmern Konkurrenz macht.

Ende August 2011 ordnet das Gericht an, dass Google die Lindholm-E-Mails an Oracle herausgeben muss. Google kämpft allerdings weiter darum, die Mail zurückzuhalten – vielleicht schon in Vorbereitung einer möglichen Berufungsverhandlung. Die schlechten Nachrichten für Google gehen weiter: Nachdem Richter Alsup bereits unter anderem Googles Forderung abgeschmettert hatte, kostenloser Software müsse eine mittelbare Patentverletzung gestattet sein, hat er nun auch den Antrag abgelehnt, über potenzielle Urheberrechtsverletzungen als „summary judgment" zu urteilen, vor der eigentlichen Verhandlung vor einer Jury. Auch die bisherigen Vernehmungen von Google-Ingenieuren wie Joshua Bloch zu den Urheberrechtsverletzungen, in denen auch (mehr als) verdächtiger Source-Code präsentiert wird, laufen nicht gut für Google.

Am 19. September 2011 erscheinen Larry Ellison und Larry Page auf Anordnung des Gerichts zu einem Mediations-Termin. Eine außergerichtliche Einigung scheint jedoch in weiter Ferne zu liegen. Ein zweiter und dritter Termin werden vereinbart, jedoch ohne die beiden CEOs – vermutlich ein Hinweis darauf, dass niemand wirklich an eine Einigung glaubt.

Das Risiko für Google, es auf ein Gerichtsverfahren ankommen zu lassen, scheint dabei recht hoch zu sein. Florian Müller beschreibt in seinem Blog, wie der gerichtliche Entscheidungsprozess aussehen würde und welche Summen letztlich im Spiel wären. Der schlimmste Fall für Google wäre, nicht nur der vorsätzlichen Verletzung von Patenten und Urheberrechten in der Vergangenheit schuldig gesprochen zu werden, sondern aufgrund einer gerichtlichen Verfügung auch in Zukunft weitere Verletzungen unterlassen zu müssen – ohne die Möglichkeit, dies durch Änderungen an Android zu umgehen. In diesem Fall müsste Google sich mit Oracle einigen. Die Forderungen könnten dann laut Müller mehrere Milliarden Dollar pro Jahr betragen. Die Berechnung basiert auf den Annahmen einer weiterhin florierenden Android-Plattform und einer Lizenzabgabe von 15 bis 20 Dollar pro Gerät an Oracle.

Ende September 2011 werden beide Parteien vom Gericht gebeten, sich zu einer möglichen Aussetzung des Verfahrens (oder Teilen davon) zu äußern, bis das US-Patentamt eine endgültige Entscheidung bezüglich der von Google angefochtenen Patente getroffen hat. Wie erwartet möchte Oracle so schnell wie möglich mit der Hauptverhandlung beginnen, Google hingegen möchte das gesamte Verfahren aussetzen (und könnte so Zeit gewinnen, um unter anderem den Kauf von Motorola Mobility abzuschließen). Eine mögliche Variante wäre, zunächst nur über die Urheberrechtsverletzungen und gegebenenfalls einen kleinen Teil der Patente zu verhandeln. Das würde allerdings ein weiteres Verfahren für den Rest bedeuten – wenn sich die Parteien unter dem Druck des ersten Verfahrens nicht doch noch rechtzeitig einigen: Ein Pokerspiel für den Richter, der letztlich die Entscheidung trifft.

Unabhängig von der Entscheidung über eine Aussetzung kündigt das Gericht wenige Tage später an, dass sich der Beginn der Hauptverhandlung aufgrund von Termin-Konflikten mit einem anderen Verfahren vermutlich verzögern wird.

Trotzdem gehen die Vorbereitungen im Oktober weiter: Oracle und Google stellen ihre Zeugenlisten zusammen, auf denen natürlich viele bekannte Namen auftauchen (neben den CEOs der beiden Kontrahenten z.B. etwa Scott McNealy und James Gosling und natürlich die bereits erwähnten Andy Rubin und Tim Lindholm). Daneben versuchen sie, jeweils für sie ungünstige Punkte noch aus der Hauptverhandlung auszuschließen und bringen ihre Argumente in den „trial briefs" an das Gericht noch einmal in Stellung (siehe http://fosspatents.blogspot.com/2011/10/trial-briefs-filed-oracle-wants.html). Für Oracles Argumentation spielt dabei die Frage der „Fragmentierung" des Java-Ökosystems durch Android eine zentrale Rolle. Ende Oktober 2011 kehrt dann Ernüchterung ein für alle, die auf ein schnelles Ende des Verfahrens hoffen: Richter Alsup verschiebt die Hauptverhandlung auf 2012. Dann soll das Verfahren in drei Phasen durchgeführt werden: Zunächst soll es um die mögliche Verletzung des Urheberrechts gehen, dann um die Patente und schließlich um die Frage der Vorsätzlichkeit und die Bezifferung der Schadenshöhe – alles vor derselben Jury. Richter Alsup hat dies explizit mit der Hoffnung verbunden, dass beide Parteien nach Abschluss der ersten Phase „schon" eine Einigung erreichen (was im Falle eines für Google günstigen Ausgangs dieser Phase aber wohl unwahrscheinlich wäre). Mindestens für einige Monate wird uns diese Hängepartie aber noch erhalten bleiben – an eine Einigung noch vor der Hauptverhandlung glaubt momentan sicher kaum noch jemand.

Wer sich für mehr Details zur Android-Klage und ähnliche Themen interessiert, dem sei das Blog von Florian Müller ans Herz gelegt, das nicht nur für den vorliegenden Artikel, sondern auch für Zeitungen und Magazine in aller Welt eine unschätzbare Informationsquelle ist: http://fosspatents.blogspot.com

Andreas Badelt ist Berater und Softwareentwickler / -Architekt bei Infosys Limited. Daneben organisiert er seit 2001 die Special Interest Group (SIG) Development bzw. SIG Java der DOAG.

Android: von Aktivitäten und Absichtserklärungen

Andreas Flügge, Object Systems GmbH

Das Application-Framework von Android ist die Basis für alle Anwendungen. Es führt eine Reihe von Konzepten ein, mit denen sich jeder Entwickler vertraut machen muss, um Android-Applikationen effektiv implementieren zu können. Der Artikel erläutert kurz die wichtigsten Aspekte des Frameworks. Außerdem ist dargestellt, wie man ein Grundgerüst als Grundlage zur Entwicklung eigener Applikationen erstellt.

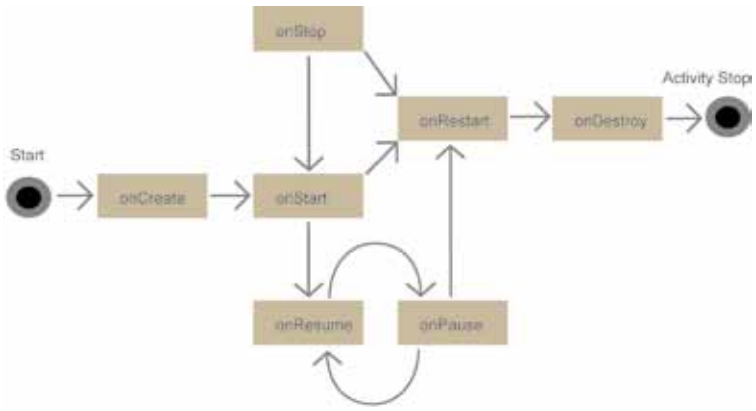
Android hat eine streng Komponenten-orientierte Architektur. Das Application-Framework, auf dem alle Android-Applikationen basieren, führt eine Reihe von Konzepten ein, die für den Entwickler von grundlegender Bedeutung sind.

Ohne ein Verständnis dieser Konzepte ist eine effektive Entwicklung von Android-Applikationen nicht möglich, weshalb die wichtigsten nachfolgend kurz erläutert werden:

- „Activities" sind ein Konzept für die Benutzeroberfläche. Sie repräsentieren normalerweise einen einzelnen Screen einer Applikation und bestehen aus einer oder mehreren Views. Es kann in besonderen Fällen auch Activities ohne Benutzeroberfläche geben. Sie haben einen ganz speziellen Lebenszyklus, der durch das Laufzeitsystem von Android gesteuert wird.
- „Views" sind hierarchisch angeordnete Bausteine, aus denen sich die Benutzeroberfläche einer Applikation zusammensetzt. Das können Buttons, Labels, Textfelder oder andere, komplexere Elemente sein.
- „Intents" sind Absichtserklärungen, um eine bestimmte Aufgabe zu übernehmen. Ein Intent kann eine Activity aufrufen, eine Nachricht versenden, eine Webseite darstellen oder eine Telefonnummer wählen. Komponenten, die Kenntnisse über Intents besitzen, können diese dazu nutzen, Aufgaben ausführen zu lassen. Intents stellen das zentrale Element zur losen Koppelung und zur Wiederverwendung bereits vorhandener Komponenten dar.
- „Content Provider" sind Komponenten, die Daten mehreren anderen Komponenten zur Verfügung stellen. Ein typisches Beispiel dafür ist das Adressbuch, das die Kontakte für andere Anwendungen auf einem Gerät zugänglich macht.
- „Services" sind Dienste, die lang laufende Aufgaben als Hintergrundprozesse ohne direkte Benutzerschnittstelle erledigen. Sie werden als „lokal" bezeichnet, sofern sie nur der implementierenden Anwendung zur Verfügung stehen. Werden sie auch von anderen Anwendungen des Gerätes genutzt, bezeichnet man sie als „Remote Service".
- „Broadcast Receiver" empfangen Systemnachrichten. Sie dienen dem Erkennen und Verarbeiten von Systemereignissen wie dem Abbruch einer Netzwerkverbindung, dem Eintreffen eines Anrufs oder der Warnung vor einer geringen Akkuleistung.

Application Lifecycle

Android behält die Kontrolle über den Lebenszyklus aller Ressourcen des laufenden Systems. Das betrifft sowohl die Prozesse als auch die Komponenten, aus denen eine Android-Applikation besteht. Diese Verwaltung basiert auf einem dynamischen Priorisierungssystem, das beispielsweise dafür Sorge trägt, dass ein im Hintergrund laufender Service einen eintreffenden Telefonanruf nicht blockieren kann.



Die Android-Architektur

Eine Applikation wird nur beim ersten Start in den Hauptspeicher geladen. Wird sie beendet, verbleibt sie im Speicher, um beim erneuten Start schneller zur Verfügung zu stehen. Erst wenn die Ressourcen knapp werden, werden inaktive Applikationen wieder entfernt. Weiter sind nur Applikationen, die sich gerade im Vordergrund befinden (angezeigt werden), aktiv. Wird eine aktive Applikation durch eine andere ganz oder teilweise überdeckt, wird die aktive Anwendung angehalten oder abgebrochen.

Entwickler müssen diesen Lebenszyklus genau kennen, um stabile Anwendungen entwickeln zu können. Denn nur dann, wenn die Applikation korrekt auf die Ereignisse reagiert, sind Datenverlust oder unnötiger Energieverbrauch zu verhindern. Das Application Framework stellt dazu eine Reihe von Methoden zur Verfügung, die bei den entsprechenden Ereignissen automatisch aufgerufen werden. Was das im Detail bedeutet, soll im Folgenden am Lifecycle der Activity erläutert werden (siehe Abbildung 1).

Typischerweise wird die Methode „onCreate“ dazu verwendet, wichtige Initialisierungen durchzuführen und benötigte Ressourcen zu reservieren. Die Methoden „onPause“ und „onStop“ können beispielsweise dazu verwendet werden, vorübergehend nicht benötigte, Energie-intensive Funktionen (wie GPS-Empfänger) abzuschalten, um sie in den Methoden „onResume“ oder „onRestart“ wieder zu aktivieren. Zum Schluss kommt die Methode „onDestroy“ zum Einsatz, um alle vorher reservierten Ressourcen wieder freizugeben.

Oberflächliches

Der Benutzeroberfläche kommt auf mobilen Geräten eine besondere Bedeutung zu. Unterschiedliche Display-Auflösungen und das Thema „Internationalisierung“ (Sprachen, Zeiten, Währungen, Einheiten) sind nur zwei Aspekte, die an dieser Stelle stellvertretend genannt werden. Android trägt diesen Aspekten unter anderem durch sein Konzept der Layouts und Ressourcen Rechnung. Es kann durch dynamische Layout-Applikationen in die Lage versetzen, sich unterschiedlichen Bildschirmauflösungen – einschließlich der Umschaltung zwischen Hoch- und Querformat – anzupassen. Ressourcen (Bilder, Videos etc.) können skaliert oder in passenden Auflösungen zur Verfügung stehen, die automatisch zur Darstellung ausgewählt werden. Views und Layouts werden normalerweise in XML-Dateien beschrieben.

Jede Android-Applikation ist standardmäßig mehrsprachig. Text-Ressourcen werden über IDs angesprochen, die in mehreren Sprachversionen vorliegen können. Unterstützt die Applikation nicht die vom Gerät benutzte Sprache, so werden die Texte der Default-Sprache verwendet.

Sicherheit

Neben dem Sandbox-Prinzip, das für jede Android Anwendung getrennte virtuelle Maschinen, Prozesse und Dateisysteme vorsieht, existieren in Android noch eine Reihe weiterer Sicherheitskonzepte, zum Beispiel die Berechtigungen. Android definiert etwa hundert unterschiedliche Berechtigungen. Dahinter verbergen sich sicherheitsrelevante Aspekte, die eine Anwendung nur mit Zustimmung verwenden darf. Dazu gehören beispielsweise die Verwendung einer Internetverbindung, der Zugriff auf die Telefonfunktionalitäten oder bestimmte Administrationsrechte.

Um die Berechtigungen zu erlangen, muss eine Applikation diese im Manifest deklarieren. Bei der Installation der Anwendung wird dem Benutzer eine Liste der angeforderten Berechtigungen angezeigt. Akzeptiert er diese nicht, wird die Installation abgebrochen. Eine Anwendung, die versucht, von sicherheitsrelevanten Funktionen Gebrauch zu machen, ohne diese zu deklarieren, wird durch die Laufzeitumgebung von Android mit einer Exception beendet.

Android Manifest

Damit eine Android-Applikation auf einem Gerät oder dem Emulator eingesetzt (deploy) werden kann, ist das sogenannte „Android Manifest“ erforderlich. Dabei handelt es sich um eine XML-Datei, die mit den Deployment-Deskriptoren aus JEE-Anwendungen vergleichbar ist. Das Manifest befindet sich im Wurzelverzeichnis der Applikation und enthält unter anderem folgende Informationen: Package-Name, Version und API-Level der Anwendung, verwendete Bibliotheken, alle Komponenten (Activities, Content Provider, Services und Broadcast Receiver) und Berechtigungen.

Applikationsgerüst

Der im SDK enthaltene Wizard ist in der Lage, ein lauffähiges Grundgerüst einer Android-Applikation zu generieren. Er benötigt dabei nur einige Informationen über die Art der zu erstellenden Anwendung. Nehmen wir an, wir wollen eine Applikation erstellen, die mithilfe von Google Maps unsere aktuelle Position darstellen soll. Damit lässt sich zum einen demonstrieren, wie eine Funktionalität des Device (der GPS-Empfänger) verwendet wird, und zum anderen, wie man auf bereits vorhandene Dienste und Libraries zugreift (Google Maps).

Wir gehen davon aus, dass Eclipse und das Android-SDK inklusive Plug-in bereits installiert sind. Eine genaue Beschreibung der Installation befindet sich unter der Downloadseite des Android-Developers (siehe http://developer.android.com/sdk/index.html).

Nachdem wir Eclipse geöffnet haben, legen wir zunächst ein neues virtuelles Device an, für das wir unsere Applikation entwickeln möchten. Das geschieht über den „Android SDK and AVD-Manager“ (siehe Abbildung 2, fünfter Button von links). Die Einstellungen nehmen wir anhand von Abbildung 3 vor. Normalerweise würde man hier eine ganze Reihe von virtuellen Geräten anlegen, um möglichst viele verschiedene Geräte in unterschiedlichen Konfigurationen testen zu können.

Abbildung 2: Buttons des Android-Plug-ins

Als Target wählen wir die Google APIs mit Level 8, da wir diese für die Map-Darstellung benötigen. Der API Level 8 entspricht einem Android 2.2 (Froyo). Mit der Skin-Einstellung legen wir unter anderem die Auflösung des Displays unseres Test-

Abbildung 3: Konfiguration des virtuellen Device

Device fest, hier 240 x 400 Pixel. Sollte die Auswahl nicht zur Verfügung stehen, sind die entsprechenden Komponenten über „Available packages" nachzuinstallieren. Das Ergebnis sollte dann wie in Abbildung 4 dargestellt aussehen.

Jetzt legen wir ein neues Android-Projekt an. Dazu verwenden wir den Android-Wizard (siehe Abbildung 2, sechster Button von links). Den folgenden Dialog füllen wir aus, wie in Abbildung 5 zu sehen ist, und beenden ihn über den „Finish"-Button.

Mit den Eingaben des Dialogs haben wir den Projektnamen, die Paketstruktur und den Namen der Start-Activity festgelegt. Der Eintrag „Min SDK Version" legt fest, dass unsere Applikation nur auf Geräten läuft, die mindestens mit Android 2.2 bestückt sind. Das hat hier keine technische Notwendigkeit und soll nur als Beispiel dienen. Als Resultat erhalten wir die Projektstruktur (siehe Abbildung 6).

Der Wizard hat eine vollständige Android-Applikation erstellt, die wir sofort im virtuellen Device testen können. Wenn wir im Kontextmenü von MyLocation „Run As" und „Android Application" auswählen, wird der Emulator mit dem konfigurierten TestDevice und unserer Anwendung gestartet. Das Ergebnis sieht (nach eventueller Entriegelung des Device) noch etwas unspektakulär aus. Die eigentliche Funktionalität muss jetzt noch in das Gerüst eingefügt werden.

Fazit

In diesem Artikel haben wir die wichtigsten Konzepte des Application Framework erklärt, uns mit dem Lifecycle einer Activity beschäftigt und das Gerüst einer lauffähigen Android-Applikation mit dem Wizard erstellt. Damit sind die Grundlagen zur Erstellung eigener Applikationen gelegt. Weitere Informationen zum Framework und zur Detail-Implementierung eigener Funktionalitäten stehen im Android Developer's Guide (http://developer.android.com/guide/index.html).

Andreas Flügge
info@ordix.de

Andreas Flügge ist seit 2002 Senior Consultant bei der Object Systems GmbH, einer Tochtergesellschaft der ORDIX AG. Seit 1999 ist er im Java-Umfeld tätig. Seine Schwerpunkte sind Java-Enterprise-Umgebungen und Software-Architekturen.

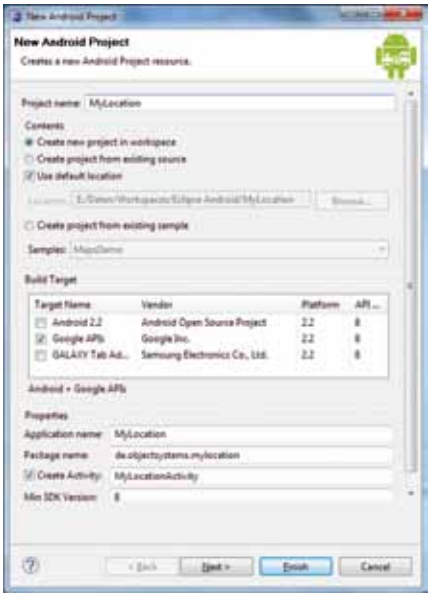


Abbildung 5: Projekt-Wizard

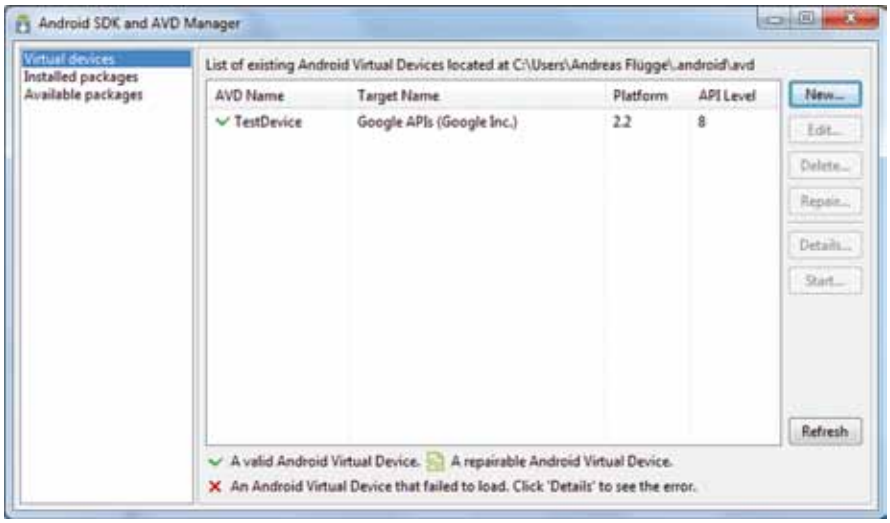


Abbildung 4: AVD-Manager nach Anlegen des TestDevice

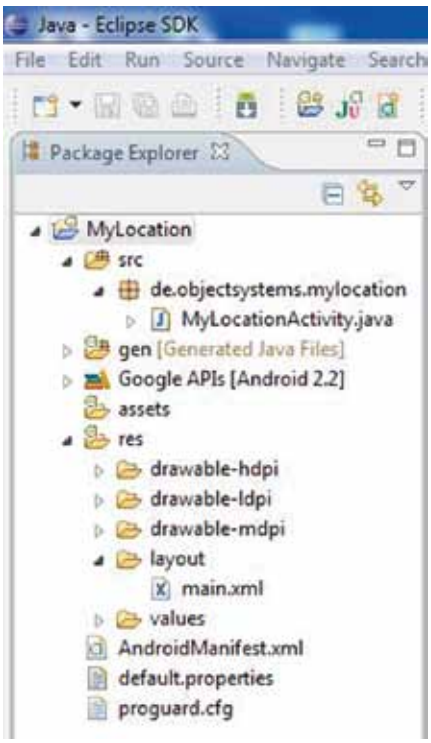


Abbildung 6: Projektstruktur

Zusammengesetzte Persistenz-Einheiten

Bernd Müller, Ostfalia – Hochschule für angewandte Wissenschaften, und Harald Wehr, MAN Truck & Bus AG

Die Persistence-Unit ist eines der zentralen Konzepte der Java-Persistence-API. Sie wird in Form der Datei „persistence.xml" verwendet, um dem JPA-Provider notwendige Informationen zur Verwaltung von Entities zur Verfügung zu stellen, etwa die zu verwendende Datenbank inklusive deren Verbindungs- und Authentifizierungsdaten. Seit der Version 2.3 erlaubt der JPA-Provider „EclipseLink" die Verwendung einer sogenannten „Composite-Persistence-Unit". Diese stellt eine logische Persistence-Unit dar, die aus mehreren Teilen, sogenannten „Composite-Member-Persistence-Units" besteht. Die Verwendung erfolgt für den Entwickler zu großen Teilen transparent, geht aber an dieser Stelle auch über den JPA-Standard hinaus.

Eine Instanz des Interface „EntityManagerFactory" repräsentiert zur Laufzeit eine Persistence-Unit. Die Verwendung mehrerer Persistence-Units in einer JPA-Anwendung ist möglich. Diese werden durch mehrere Instanzen des Interface „EntityManagerFactory" unterschieden. Eine Instanz des Interface „EntityManager" repräsentiert einen Persistence-Context, eine Menge von Entity-Instanzen. In Java SE werden „EntityManagerFactory" und „EntityManager" programmatisch erzeugt, in Java EE durch Verwendung der Annotationen „@PersistenceUnit" und „@PersistenceContext" vom Container injiziert.

Durch die Verwendung der EclipseLink-Erweiterung der Composite-Persistence-Units entfällt die explizite, programmatische Verwaltung der EntityManager und EntityManager-Fabriken. Dies übernimmt EclipseLink. Wir entwickeln im Folgenden eine kleine Anwendung zur Verwaltung der Abteilungen einer Firma sowie deren Mitarbeiter und ihrer Adressen, also das typische Department-Employee-Beispiel.

Die Fachlichkeit

Die Fachlichkeit besteht aus den Entities „Mitarbeiter", „Abteilung" und „Adresse", wobei ein Mitarbeiter zu einer Abteilung gehört (n:1) und eine Adresse besitzt (1:1). Wir geben die wichtigen Ausschnitte der Klassen wieder (siehe Listing 1).

Abbildung 1 zeigt das Datenbank-Schema für die drei Entity-Klassen. Wir haben die Zuordnung zu den beiden Composite-Member-Persistence-Units, die im Folgenden eingeführt werden, bereits vorweggenommen.

Composite-Member-Persistence-Unit

Die Composite-Member-Persistence-Units werden wie gewöhnliche Persistence-Units verwendet, um Datenquellen zu definieren. In unserem Beispiel sollen die Klasse „Mitarbeiter" in einer, die Klassen „Abteilung" und „Adresse" in einer anderen Composite-Member-Persistence-Unit enthalten sein. Die erste Persistence-Unit für die Klasse „Mitarbeiter" stellt sich wie folgt dar (siehe Listing 2).

Man erkennt den Namen der Persistence-Unit, hier „employee1", die Verwendung einer H2-Datenbank und das Setzen des Properties „eclipselink.composite-unit.member" auf „true". Damit wird gekennzeichnet, dass diese Persistence-Unit nur in einer Composite-Persistence-Unit und nicht stand-alone verwendet werden kann. Der Grund hierfür liegt in den beiden Assoziationen der Klasse „Mitarbeiter": Die Persistence-Unit kann nur mit den entsprechenden Persistence-Units der Assoziationsziele und nicht allein verwendet werden.

Die zweite Composite-Member-Persistence-Unit „employee2" ist analog aufgebaut und verwendet eine PostgreSQL-Datenbank. Da die beiden Klassen „Abteilung" und „Adresse" keine Assoziationsziele in anderen Persistence-Units besitzen, könnte sie auch stand-alone verwendet werden, sodass auf das entsprechende Property verzichtet werden kann (siehe Listing 3).

```
@Entity
public class Mitarbeiter {
  @Id @GeneratedValue
  private Integer id;
  private String vorname;
  private String nachname;
  @Temporal(TemporalType.DATE)
  private Date geburtsdatum;
  @OneToOne
  private Adresse adresse;
  @ManyToOne
  private Abteilung abteilung;
  ...
}

@Entity
public class Adresse {
  @Id @GeneratedValue
  private Integer id;
  private String strasse;
  private String hausnummer;
  private String plz;
  private String ort;
  ...
}

@Entity
public class Abteilung {
  @Id @GeneratedValue
  private Integer id;
  private String name;
  ...
}
```

Listing 1

Packaging

Beim Erzeugen der Jars wird die erste Composite-Member-Persistence-Unit ent-

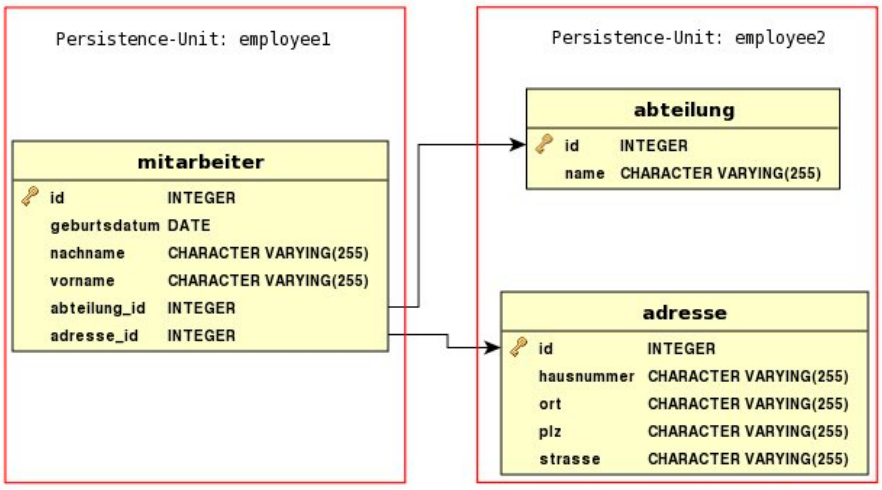


Abbildung 1: Datenbankschema für die Entity-Klassen „Mitarbeiter", „Abteilung" und „Adresse"

```
<persistence ...>
  <persistence-unit name="employee1">
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver"
              value="org.h2.Driver" />
      <property name="javax.persistence.jdbc.url"
              value="jdbc:h2:tcp://localhost/bank" />
      <property name="javax.persistence.jdbc.user" .../>
      <property name="javax.persistence.jdbc.password" .../>
      <property name="eclipselink.composite-unit.member"
              value="true"/>
    ...
</persistence>
```

Listing 2

```
<persistence ...>
  <persistence-unit name="employee2">
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver"
              value="org.postgresql.Driver" />
      <property name="javax.persistence.jdbc.url"
              value="jdbc:postgresql://localhost/bank" />
      <property name="javax.persistence.jdbc.user" .../>
      <property name="javax.persistence.jdbc.password" .../>
    ...
</persistence>
```

Listing 3

```
<persistence >
  <persistence-unit name="composite">
    <jar-file>member1.jar</jar-file>
    <jar-file>member2.jar</jar-file>
    <properties>
      <property name="eclipselink.composite-unit"
              value="true" />
    ...
</persistence>
```

Listing 4

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory
    („composite“);
```

Listing 5

```
Mitarbeiter mitarbeiter = new Mitarbeiter(...);
Abteilung abteilung = new Abteilung(...);
Adresse adresse = new Adresse(...);
mitarbeiter.setAbteilung(abteilung);
mitarbeiter.setAdresse(adresse);
// em injiziert oder über Fabrik erzeugt:
em.persist(mitarbeiter);
```

Listing 6

sprechend den JPA-Regeln im Verzeichnis „META-INF“ als „persistence.xml“ gepackt. Zusätzlich ist die Anwendungsklasse „Mitarbeiter“ im Jar enthalten. Die zweite Composite-Member-Persistence-Unit wird analog mit den Klassen „Abteilung“ und „Adresse“ in ein weiteres Jar gepackt. Wir verwenden die Namen „member1.jar“ und „member2.jar“. Diese beiden Jars werden in der Composite-Persistence-Unit verwendet. Das Packaging selbst erfolgt sinnvollerweise mit Ant. Eclipse oder Maven können nicht verwendet werden, da das Packaging nicht den Standardregeln entspricht.

Composite-Persistence-Unit

Die Composite-Persistence-Unit referenziert die beiden Jars über das Standard-JPA-Tag „<jar-file>“. Wichtig ist das Property „eclipselink.composite-unit“, das auf „true“ gesetzt werden muss, da es sich um eine zusammengesetzte Persistenzeinheit handelt. Die Composite-Persistence-Unit enthält keine Verbindungsinformationen für eine Datenquelle. Wird eine Datenquelle in einer Composite-Persistence-Unit definiert, so wird dies ignoriert (siehe Listing 4).

Verwendung

Die „EntityManagerFactory“ wird nun, wie in JPA üblich, über die entsprechende Create-Methode erzeugt (siehe Listing 5).

Beim Persistieren von Entities kann völlig transparent mit den drei Entity-Klassen gearbeitet werden. Wenn bei den Annotationen „@ManyToOne“ und „@OneToOne“ der „CascadeType“ entsprechend gesetzt wird, werden mit dem folgenden Code sowohl der Mitarbeiter als auch seine Abteilung und seine Adresse persistiert (siehe Listing 6). Der Mitarbeiter landet in der H2-Datenbank, Abteilung und Adresse in der PostgreSQL-Datenbank. Beim Lesen eines Mitarbeiters mit „em.find()“ werden die beiden Assoziationen korrekt initialisiert, es findet also ein Lesen in beiden Datenbanken statt.

Beschränkungen

Die Verteilung von Entities auf verschiedene Persistenzeinheiten schränkt die Möglichkeiten, die der JPA-Standard bietet, in bestimmten Bereichen stark ein:

- Queries mit Entities verschiedener Composite-Member-Persistence-Units sind nicht erlaubt
- Vererbungshierarchien können nicht auf mehrere Composite-Member-Persistence-Units aufgeteilt werden
- Bei der Verwendung von „@JoinTable“ für 1:n- und n:m-Beziehungen muss die Join-Tabelle in derselben Persistence-Unit wie das Beziehungsziel sein und die Verwendung von „mappedBy“ ist nicht zulässig. Daher können derartige Beziehungen nicht bidirektional sein.

Es gibt noch eine Reihe weiterer Einschränkungen, die wir aber nicht im Einzelnen aufführen. Wir verweisen den Leser auf das EclipseLink-Wiki.

Fazit

Die genannten Beschränkungen reduzieren den allgemeinen Einsatz von Composite-Persistence-Units erheblich, da grundlegende JPA-Mechanismen nicht mehr verwendet werden können. Wir sind dennoch der Meinung, dass sich der Einsatz lohnen kann.

In großen Anwendungen ergibt sich in der Regel ganz automatisch eine Partitionierung von Entity-Klassen, sodass keine Assoziationen und keine gemeinsame Vererbungshierarchie über Partitionsgrenzen hinweg existieren. Die genannten Beschränkungen treffen in diesem Fall dann nicht zu.

Außerdem kann es aus Effizienz- und Kostengründen sinnvoll sein, sich häufig ändernde Daten, wie normale betriebliche Anwendungsdaten, etwa in Oracle zu halten, während sehr viel umfangreichere, binäre Daten, wie etwa monatliche Abrechnungen, in PDF-Form in dafür geeigneteren Datenbanken abgelegt werden können. Ein weiterer Anwendungsfall wäre, die Ablage von datenschutzrelevanten Informationen in besonders geschützten Security-Zonen beziehungsweise Datenbanken durchzuführen und strikt von weniger sensiblen Daten zu trennen. Auch die Zusammenführung von Informationen aus verschiedenen existierenden Datenbanken beispielsweise für Auswertungen ist mit der EclipseLink-Erweiterung problemlos möglich. Daten aus unterschiedlichen Unternehmensbereichen oder Datenbanken, die fachlich Bezug zueinander haben, können so mit einem übergreifenden Business-Modell schnell zusammengeführt werden. Effiziente SQL-Befehle und deren JPA-Entsprechung bleiben jedoch außen vor, da man mit verschiedenen Datenbanken arbeitet.

Weiterführende Informationen

[1] http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Composite_Persistence_Units

[2] http://wiki.eclipse.org/EclipseLink/Examples/JPA/Composite

Bernd Müller
bernd.mueller@ostfalia.de

Harald Wehr
harald.wehr@gmail.com

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

Harald Wehr ist seit 2005 bei der MAN Gruppe in Salzgitter als Java-Entwickler beschäftigt. Seit 2008 befasst er sich verstärkt mit der SAP HR Anwendungsentwicklung und Beratung. Zusammen mit Bernd Müller verfasste er das Buch „Java-Persistence-API mit Hibernate“. Die Autoren arbeiten derzeit an einer Neuauflage des Buches, das im kommenden Frühjahr beim Hanser-Verlag erscheinen wird.

Java und HPC: Wirklichkeit oder Widerspruch?

Johannes M. Dieterich, Georg-August-Universität Göttingen

High performance computing (HPC) spielt eine herausragende Rolle in den Natur- und Ingenieurswissenschaften. Dieser Artikel soll die speziellen Anforderungen dieses Fachgebiets aus Entwicklerperspektive darstellen und auf ihre Vereinbarkeit mit der Java-Plattform hin untersuchen.

Java ist langsam. Vermutlich jedem ist diese spontane Einschätzung schon begegnet. Dieser Logik folgend eignet sich Java selbstverständlich nicht als Programmiersprache für Anwendungen im High Performance Computing (HPC). Doch wie gut lässt sich dieses Urteil begründen? Ist es eine anerkannte Tatsache oder nur ein Vorurteil? Eine endgültige Antwort auf diese Frage ist naturgemäß unmöglich, denn selbst innerhalb der engen HPC-Sparte können sich Anforderungsprofile drastisch unterscheiden. Daher bleibt es jedem Leser überlassen, welchen Schluss er final für sich und sein Projekt zieht. Zu diesem Zweck wird nach einer kurzen Bestandsaufnahme und Skizze der Historie von Java im Kontext von HPC detailliert auf die Anforderungen eingegangen werden, die HPC typischerweise an das komplette Entwicklungsökosystem stellt. Verbunden mit einem Abgleich darüber, wo Java diesen Anforderungen genügt und wo weiterhin Entwicklungsbedarf besteht, soll ein möglicher Weg in die Zukunft für Java und HPC beleuchtet werden.

Historie und Gegenwart

Die Idee, Java für HPC-Anwendungen zu verwenden, ist nicht neu. Schon 1998 wurde das sogenannte „Java Grande Forum" (JGF) in Palo Alto gegründet. Es sollte die Verwendung von Java für Programme der Grande Klasse (Ressourcen-intensive HPC-Anwendungen mit Fokus auf Probleme der Natur- und Ingenieurswissenschaften) propagieren und zentrale Anlaufstelle für Benchmarks und Verbesserungen von Java in diesem Bereich sein. Nach mehreren Konferenzen ist dieses Forum seit 2003 nicht mehr aktiv [JavaGrande]. Bemerkenswert ist, dass die damals unternommenen Anstrengungen (Entwicklung einer Benchmark-Suite, Optimierung der Parallelisierung, gezielte Sprachänderungen) heute genauso aktuell wie damals sind. Auch sei an dieser Stelle auf die aus diesem Umfeld entstandene, exzellente Studie zum Thema von Smith und Bull von 2001 verwiesen [JavaAndHPC], die trotz inzwischen veralteter Ergebnisse Maßstäbe in Bezug auf Darstellung und Vollständigkeit der Daten setzt.

Aus naturwissenschaftlicher Perspektive hat sich allerdings die HPC-Welt seitdem auch nicht von Grund auf gewandelt. Die Maschinen sind mächtiger geworden, die darauf laufenden Codes sind ungefähr gleich geblieben. Gegenwärtiger Standard sind weiterhin Fortran-basierte Programme, die typischerweise in Fortran77- oder Fortran95-Mischformen geschrieben sind. C/C++-basierte Programme stellen bisher (zumindest in den Naturwissenschaften) nur eine kleine Minorität, wobei bei neuen Programmen die Verwendung von C/C++ wahrscheinlicher geworden ist. Andere Sprachen wie Java spielen nahezu keine Rolle.

Für die Dominanz von Fortran gibt es mehrere Gründe. Der wichtigste Grund ist sicherlich, dass Fortran als Sprache für numerische Anwendungen eine jahrzehntelange Tradition hat. Hieraus resultieren eine große Menge an Legacy-Codes und eine breite Akzeptanz in der Community. Bei Weiterentwicklungen oder neuen Projekten ist Fortran als Standard gesetzt, jede andere Wahl muss begründet werden. Auch ist zumindest an großen deutschen Rechenzentren Support bei der Portierung von Codes auf die jeweilige Architektur nur bei Fortran gesichert.

Anforderungen von HPC-Anwendungen an ein Sprach-Ökosystem

Neben dem Offensichtlichen – schnellem Programmcode – spielen auch noch andere Faktoren eine wichtige Rolle. Genannt sei architekturübergreifende, numerische Stabilität. Eine Implementation muss auf verschiedenen Hardware-Konfigurationen reproduzierbar das gleiche Ergebnis liefern.

Möglichkeiten der Parallelisierung von Programmen sind nahezu von Beginn an im HPC von größter Wichtigkeit gewesen. Darüber hinaus muss die Verfügbarkeit von numerischen Bibliotheken (etwa BLAS/LAPACK) gegeben sein. Ob eine Sprache objektorientiert ist oder nicht, spielt in den harten numerischen Subroutinen kaum eine Rolle (trotz Werken auf diesem Gebiet [OONumerics]). Die Sprache muss außerdem für HPC-Zwecke einfach zu verwenden sein. Letzteres ist einer der Punkte, die bisher dem Durchbruch von C/C++ im Wege standen. Der Hintergrund ist einfach: viel HPC-Entwicklung wird von Naturwissenschaftlern und Ingenieuren auf ihren jeweiligen Gebieten selbst geleistet. Naturgemäß ist deren Kenntnisstand auf dem Gebiet der Softwareentwicklung stark fokussiert. Primärziel ist die Lösung eines definierten Problems mit einem funktionalen Werkzeug, nicht die Herstellung desselben.

Nutzbarkeit und Portabilität

Java ist als plattformunabhängige Sprache höchstportabel und garantiert eine definierte Genauigkeit numerischer Berechnungen unabhängig von Architektur und Hardware. Darüber hinaus ist sie im Vergleich zu C/C++ deutlich einfacher (besonders durch die automatische Speicherverwaltung) und weniger fehleranfällig zu verwenden. Das Gesamt-Ökosystem der Java-Umgebung mit gut benutzbaren Profilern und Debuggern lernt man besonders im Vergleich zur sehr unkomfortablen Situation bei Fortran77 zu schätzen. Fehler zur Laufzeit werden detaillierter gemeldet und weniger verschleiert. Die prinzipielle Überlegenheit von objektorientierten Sprachen für nicht-triviale Programm-Architekturen sei nur am Rande erwähnt. Hieraus resultiert klassisch eine höhere Funktionalität von objektorientierten Programmen im Vergleich zu prozeduralen Sprachen wie Fortran bei gleichem Codeumfang. Diese Metrik ist auch bei HPC-Programmen von großer Bedeutung.

Hervorzuheben ist, dass auch im HPC neue Algorithmen stärkeren Gebrauch von modernen Datenstrukturen (Hashtabellen, Baumstrukturen etc.) und Spracheigenschaften (wie Stringoperationen) machen. Die Implementation dieser Algorithmen ist folglich mit modernen Sprachen wie Java deutlich einfacher als mit den Bordmitteln von Fortran77/95.

Schon kleinere Universitätsrechenzentren verfügen heute über mehrere Tausend Prozessoren. HPC-Anwendungen müssen so gut wie möglich skalieren, um diese Rechenleistung effizient zu nutzen. Traditionell spielt hierfür das Message Passing Interface (MPI) eine Rolle. Eine entsprechende, angepasste Umgebung ist an allen Rechenzentren vorhanden und kann zur Kommunikation genutzt werden. Eine Darstellung des MPI-Standards (und seiner Schwächen) würde hier den Rahmen sprengen, es sei daher nur darauf hingewiesen, dass die Verwendung einer funktionierenden, vom jeweiligen Rechenzentrum an die vorhandene Hardware und das vorhandene Queueing-System angepassten MPI-Umgebung der Königsweg ist. Dies ist auch der Grund dafür, warum die Verwendung eines JNI-basierten Interface wie mpiJava [mpiJava] einer RMI-basierten Implementation wie MPJ-Express [mpjexpress] von Rechenzentren bevorzugt wird. Aktuell wird allerdings nur das MPJExpress-Projekt weiterentwickelt und sollte daher Verwendung finden. Die Leistungsfähigkeit ist exzellent, mehrere Hundert Prozessoren lassen sich linear skalierend nutzen, falls der Algorithmus und die Implementation dies erlauben. Die Unterstützung von schnelleren Interconnects wie InfiniBand, die für HPC-Cluster typisch sind, ist allerdings noch experimentell. Allgemein wäre eine einfachere Integration in bestehende HPC-Architekturen (inklusive Queueing-System) wichtig, um Probleme mit der Systemverwaltung des Rechenzentrums zu minimieren. Die zusätzlich in Java eingebauten Parallelisierungsmöglichkeiten mittels Java-Threads oder RMI machen Java zu einer ausgesprochen mächtigen Wahl für hochskalierende HPC-Anwendungen.

```
static double[ ][ ] matMultNetlib(double[ ][ ] a, double[ ][ ] b){

    int rowsA = a.length; // Anzahl Reihen A und C
    int colsB = b[0].length; // Anzahl Spalten B und C
    int colsA = a[0].length; // Anzahl Spalten A, Reihen B
    double[ ][ ] c = new double[rowsA][colsB];
    String transA = „n“; // op(A) = A
    String transB = „n“; // op(B) = B
    double alpha = 1.0;
    double beta = 0.0;
    // C := alpha*op(A)*op(B) + beta*C
        // mit op(X) = X  oder  op(X) = X**T,
    DGEMM.DGEMM(transA, transB, rowsA, colsB, colsA, alpha, a, b, beta, c);

    return c;
}
```

Listing 1

```
static double[ ][ ] matMultJama(double[ ][ ] a, double[ ][ ] b){

    Matrix matA = new Matrix(a);
    Matrix matB = new Matrix(b);
    Matrix matC = matA.times(matB);

    return matC.getArray();
}
```

Listing 2

Bibliotheken

Von besonderer Wichtigkeit für HPC-Anwendungen ist die Verfügbarkeit von Bibliotheken mit optimiertem numerischem Code. Nahezu jedes Projekt benötigt BLAS/LAPACK-Bibliotheken für lineare Algebra. Hier ist Java mit unterschiedlichen Lösungen gut aufgestellt. Aus einer automatischen Übersetzung der Netlib entstand nicht nur eine komplette BLAS/LAPACK-Bibliothek, sondern auch ein Übersetzungstool von Fortran in Java-Bytecode [f2java]. Einziger Mangel ist, dass die Nutzung dieser Funktionen sehr archaisch ist; Listing 1 zeigt als Beispiel den Aufruf einer Matrix-Matrix-Multiplikation.

Dies kann durch Nutzung anderer Bibliotheken (die zum Teil die Netlib wrappen,) umgangen werden. Es seien Jama und UJMP als Beispiele angeführt [jama] [ujmp]. Die Nutzung dieser Pakete ist Java-typischer (hier als Beispiel Jama, siehe Listing 2).

Bei spezialisierteren Anforderungen können Funktionalitäten nicht verfügbar sein. Zwar ist es dann mö glich, über das Java Native Interface (JNI) die nativen Bibliotheken anzusprechen; der zeitliche Overhead macht diese Operation aber zur

zweiten Wahl. Es ist folglich wichtig, vor Projektstart zu bedenken, welche Bibliotheken benötigt werden und welche verfügbar sind.

Performance

Keines der angesprochenen Themen wird so intensiv und kontrovers diskutiert wie die Performance von Java, wobei typischerweise C/C++ als Vergleich genommen wird. Im HPC-Kontext ist, wie der Name „number crunching" impliziert, primär die optimale Ausnutzung der Rechenleistung von Interesse, gemessen in Fließkommaoperationen pro Sekunde (flops). Die Anzahl an Benchmarks ist hoch, die Aussagekraft leider häufig beschränkt. Nicht nur, dass sprachübergreifende Benchmarks Prinzip-bedingt nur äußerst schwer vergleichbar zu machen sind, es kommen klassische Schwächen wie mangelnde Statistik, zu kleine Problemgrößen, fehlende Informationen zur verwendeten Java-Implementation und die Verwendung von HPC-untypischen Problemklassen hinzu. Vereint mit voreingenommenen Testern (in beide Richtungen) kulminiert dieses in komplett nutzlosen Resultaten.

Eine seriösere Anlaufstelle stellt das vom Debian gehostete Computer Language Benchmark Game [LanguageShootout] dar. Die Daten und der Quellcode sind detailliert dokumentiert und es wird explizit auf die inhärenten Schwächen der Ergebnisse hingewiesen. Die hier verfügbaren Daten lassen die Schlussfolgerung zu, dass Java auch in der aktuellen 7er-Version tendenziell leicht langsamer ist als C/C++. Es sei an dieser Stelle darauf hingewiesen, dass die getesteten Problemklassen das typische Anforderungsprofil von HPC nur partiell abbilden.

HPC-typischer ist die schon genannte, vom Java Grande Forum entwickelte Benchmark-Suite [JGFBench]. Besonders in diesem Punkt wäre eine Wiederbelebung des JGF wünschenswert. Die JGF-Benchmark-Suite ist, mit kleinen Abstrichen, auch auf der aktuellsten JRE7 ohne Anpassungen lauffähig, sollte allerdings entstaubt und erweitert werden. Eine aktuell etwas besser verwendbare Benchmark-Suite für HPC-typische Anforderungen ist die SciMark-2.0-Suite [SciMark2.0]. Es sind offiziell eine Java- und eine C-Implementation verfügbar.

Tabelle 1 zeigt die Zusammenfassung eines Benchmarks mit verschiedenen C-Compilern und Flags. Sie soll lediglich einen Eindruck davon geben, wie eng die Abstände zwischen Java und C inzwischen sind, und zeigen, dass Compile-Flags einen deutlich größeren Einfluss als die Sprachwahl haben.

Einziger aktueller Benchmark für Fortran, C/C++ und Java ist Linpack [linpack]. Allerdings ist auch hier die Java-Version verstaubt und die Problemgröße muss dringend aktualisiert werden. Ein kurzer Test auf dem aktuellen i5-Notebook mit openJDK7 ergab eine Leistung von rund 50 Mflops, was einem Pentium II mit 500 MHz und JDK1.2 entsprechen würde. Eine Verzehnfachung der Matrizengrößen ergab die realistischere Größenordnung von knapp 1400 Mflops.

Da schon jetzt die Unterschiede in der Laufzeit gering sind, sich vermutlich weiter vermindern und es sicherlich berechtigte Zweifel gibt, ob Java jemals drastisch schneller sein wird als Fortran77, sollte das Augenmerk auch auf andere Zeit- beziehungsweise Performance-Komponenten gerichtet werden. Zum einen sind die Zeit-

Compiler	Flags	Normal [Mflops]	Groß [Mflops]
gcc4.2.2	-O2 -ffast-math -march=nocona	1162.8 ± 9.4	929.8 ± 1.7
	-O3 -ffast-math -march=nocona	1168.2 ± 7.5	934.8 ± 4.0
gcc4.6.2	–	450.8 ± 4.8	417.0±2.9
	O2 -ffast-math -march=corei7	1316.7 ± 5.6	986.9 ± 3.4
	-O3 -ffast-math -march=corei7	1481.6 ± 13.6	1013.5 ± 5.0
clang3.0 135360	-O2 -ffast-math -march=corei7	1367.8 ± 5.4	1040.1 ± 7.0
	-O3 -ffast-math -march=corei7	1328.3 ± 2.8	1016.4 ± 4.0
diablo-jdk6 (b07)	–	1231.0 ± 14.7	953.4 ± 9.2
openjdk7 1.7.0	–	1384.0 ± 32.7	1071.5 ± 6.0

Tabelle 1: SciMark2 auf einem FreeBSD-10-CURRENT (Intel Core i5-2520M). Alle Daten sind Composite Scores der zwei Problemgrößen gemittelt über zehn Läufe nach drei Preheating-Läufen. -march=native und -march=corei7-avx verursachten mit gcc46 einen segmentation fault

ersparnisse, die sich durch die Entwicklung und Implementation neuer Algorithmen erreichen lassen, deutlich größer als die Unterschiede zwischen den einzelnen Sprachen. Zum anderen sollte gerade in der HPC-Community der komplette Zeitbedarf eines Projekts stärkere Beachtung finden. Diese Gesamtzeit umfasst nicht nur die Laufzeit, sondern auch die Entwicklungszeit und den Portieraufwand. Zwar ist die Verwendung von vorkompilierten Programmen an Rechenzentren weit verbreitet, doch spielen Weiterentwicklungen/Optimierungen und lokale Rechenressourcen eine mindestens genauso wichtige Rolle. Für Optimierungen innerhalb von Java-Programmen sei auf das Standardwerk „Efficient Java" von Joshua Bloch verwiesen [EfficientJava]. Obwohl der Portieraufwand für ein Fortran-Programm sprachbedingt normalerweise gering ist, zeigt sich vor allem bei großen, alten Programmcodes, die Mischungen verschiedener Fortran-Dialekte und C-Anteile enthalten können, eine starke Diskrepanz zum Ideal (Java).

Fehlende Sprach-Features

In der Sprachspezifikation von Java fehlen weiterhin einige Elemente, die HPC-Anwendungen entgegenkommen würden. Operatorüberladung und Funktionspointer würden die Lesbarkeit von numerischen Codes deutlich erhöhen und die Entwicklung vereinfachen. Für einen kleineren Teil der HPC-Community wäre die Verfügbarkeit eines primitiven Complex-Datentyps, der in Anlehnung an den Fortran-COMPLEX komplexe Zahlen darstellt und Berechnungen mit ihnen erlaubt, sehr wichtig. Der Verweis, eine eigene Klasse zu verwenden, die dies implementiert, greift zu kurz, da der Overhead, ein Objekt anstelle eines primitiven Datentyps zu erzeugen, signifikant ist.
Da die Wünsche zu einer vorsichtigen Erweiterung der Java-Spezifikation in dieser Richtung schon länger bestehen und es gute Gründe gegen ein solches Unternehmen gibt, besteht hierfür vermutlich wenig Hoffnung. Umso mehr, da Sun/Oracle mit dem Projekt „Fortress" seit einigen Jahren eine JVM-basierte Sprache für numerische Anwendungen entwickelt [Fortress].

Ausblick

Viele der genannten Anforderungen werden schon jetzt von Scala erfüllt. Die optimale Integration mit bestehenden Java-Bibliotheken und -Programmen reduziert hierbei die Notwendigkeit, benötigte Bibliotheken in Scala neu zu schreiben. Sollte eine „Scalaisierung" der Bibliothek gewünscht sein, ist dies natürlich dennoch eine Option. Als Beispiel sei das Scalala-Projekt genannt [Scalala], welches eine BLAS/LAPACK-Bibliothek für Scala entwickelt. Matrizen und Vektoren lassen sich hiermit wieder natürlicher verwenden. Aus HPC-Sicht steht zu hoffen, dass der Fokus in der weiteren Sprachentwicklung von Scala mehr auf Performance und weniger auf einer weiteren Vergrößerung des Sprachumfangs liegt. Das Zusammenspiel von Java und Scala könnte in diesem Fall die Migration zu modernen Sprachen in der HPC deutlich beschleunigen (siehe Tabelle 2).

Fazit

Zusammenfassend kann gesagt werden, dass Java schon heute vielen HPC-Anforderungen genügt. Es bleibt weiterhin viel Raum für Verbesserungen, sowohl an spezialisierten und optimierten Bibliotheken als auch in der Verbreitung. Java für jegliche HPC-Anwendung prinzipiell zu verwerfen ist sicherlich nicht gerechtfertigt. Eine dem JGF ähnliche Initiative wäre auch heutzutage wünschenswert, um die weitere Verbreitung von Java im HPC zentral zu begleiten und unterstützen. Am wichtigsten für die Verbreitung von Java im HPC ist allerdings etwas anderes: selbst gute HPC-Anwendungen in Java zu schreiben.

Literatur

[JavaGrande] Java Grande Forum (JGF) - http://www.javagrande.org/

[JavaAndHPC] L. A. Smith und J. M. Bull, Java for High Performance Computing, UKHEC Technology Watch report: http://www.ukhec.ac.uk/publications/tw/hpcjava.pdf

[OONumerics] D. H. Besset, Object-oriented implementation of numerical methods, Morgan Kaufmann, 2000

[mpijava] mpiJava: http://www.hpjava.org/mpiJava.html

[mpjexpress] MPJExpress: http://mpj-express.org/

[f2java] f2j: http://icl.cs.utk.edu/f2j/

[jama] JAMA: A Java Matrix Package: http://math.nist.gov/javanumerics/jama/

[ujmp] Universal Java Matrix Package: http://sourceforge.net/projects/ujmp/

[LanguageShootout] Computer Language Benchmark Game: http://shootout.alioth.debian.org/

[JGFBench] Java Grande Benchmark: http://www2.epcc.ed.ac.uk/computing/research_activities/java_grande/index_1.html

[SciMark2.0] SciMark 2.0 - http://math.nist.gov/scimark2/

[linpack] Linpack for Java: http://www.netlib.org/benchmark/linpackjava/

[EfficientJava] J. Bloch, Effective Java 2nd edition, Addison-Wesley, 2008

[Fortress] Project Fortress: http://projectfortress.java.net/

[Scalala] Scalala: https://github.com/scalala/Scalala

Johannes Dieterich
jdieter@gwdg.de

Johannes Dieterich, Jahrgang 1985, ist seit September letzten Jahres als Postdoktorand am Institut für Physikalische Chemie der Georg-August-Universität Göttingen beschäftigt. Er befasst sich dort unter anderem mit der Weiterentwicklung genetischer Algorithmen zur globalen Optimierung chemischer Proleme. Er studierte Chemie an der Universität Stuttgart und promovierte 2010 an der Christian-Albrechts-Universität zu Kiel unter Leitung von Bernd Hartke.

	Fortran	C/C++	Java
Numerische Bibliotheken	++	+	n
Parallelisierung	+	+	++
Entwicklungsaufwand	n	-	+
Performance	+	+	+

Tabelle 2: Subjektive Zusammenfassung der HPC-relevanten Spracheigenschaften von Fortran, C/C++ und Java, n: neutral

JUnit Rules

Marc Philipp, andrena objects ag, und Stefan Birkner, Immobilien Scout GmbH

Automatisierte Tests sind aus der heutigen Softwareentwicklung nicht mehr wegzudenken. JUnit ist das älteste und bekannteste Testing-Framework für Java. Doch selbst ein so etabliertes und einfach zu benutzendes Framework wird kontinuierlich weiterentwickelt. Eine der Neuerungen sind JUnit Rules, die Entwicklern eine neue mächtige Möglichkeit bieten, Tests zu formulieren und besser zu strukturieren.

Der Legende nach haben Kent Beck und Erich Gamma 1997 den Kern von JUnit auf dem Weg zu einer Konferenz im Flugzeug zwischen Zürich und Atlanta geschrieben. JUnit griff die Idee wieder auf, die Beck 1994 mit SUnit [1] für Smalltalk eingeführt hatte: ein Testing-Framework, dessen Zielgruppe Programmierer sind, also dieselben Leute, die auch den Code schreiben, den es zu testen gilt. JUnit ist inzwischen weit verbreitet. Es wird nicht nur zum Schreiben von Unit-Tests, sondern auch zur Automatisierung von Integrations- und Akzeptanztests verwendet.

Viele erfolgreiche Open-Source-Projekte zeichnen sich dadurch aus, dass mit der Zeit immer neue Features eingebaut werden. Dies führt häufig dazu, dass einst simple Bibliotheken unübersichtlich und schwer wartbar werden. JUnit geht hier gezielt einen anderen Weg. David Saff, neben Kent Beck der zweite Maintainer von JUnit, sieht das so: „JUnit is the intersection of all possible useful Java test frameworks, not their union".

Die Wahrnehmung in der Java-Entwicklergemeinde ist dementsprechend: Da JUnit so einfach ist, meint jeder, der es schon einmal benutzt hat, es gut zu kennen. Das ist einerseits gut, denn die Hürde, Unit-Tests zu schreiben, ist so sehr niedrig. Andererseits führt es dazu, dass Neuerungen von vielen Entwicklern gar nicht oder erst verzögert wahrgenommen werden. Fragt man Entwicklerkollegen nach Neuerungen in JUnit, wird häufig die Umstellung von Vererbung auf Annotations-basierte Testschreibweise in Version 4.0 erwähnt.

```
public class TemporaryFolderWithoutRule {
    private File folder;

    @Before
    public void createTemporaryFolder() throws Exception {
        folder = File.createTempFile("myFolder", "");
        folder.delete();
        folder.mkdir();
    }

    @Test
    public void test() throws Exception {
        File file = new File(folder, "test.txt");
        file.createNewFile();
        assertTrue(file.exists());
    }

    @After
    public void deleteTemporaryFolder() {
        recursivelyDelete(folder);
    }

    private void recursivelyDelete(File file) {
        File[ ] files = file.listFiles();
        if (files != null) {
            for (File each : files) {
                recursivelyDelete(each);
            }
        }
        file.delete();
    }
}
```

Listing 1

Seitdem hat sich allerdings einiges getan. Die neueste Innovation, die mit Version 4.7 eingeführt wurde, heißt „Rules". Zugegeben, unter dem Begriff kann man sich erst einmal nichts vorstellen. Hat man sich diese „Regeln" für Tests aber einmal eingehend angesehen – und genau das werden wir in diesem Artikel tun –, stellt man fest: Rules werden die Art, wie wir JUnit-Tests schreiben, nachhaltig verändern.

Mithilfe von JUnit-Rules lässt sich die Ausführung von Tests beeinflussen. Ähn-

```
public class TemporaryFolderWithRule {

    @Rule
    public TemporaryFolder folder = new TemporaryFolder();

    @Test
    public void test() throws Exception {
        File file = folder.newFile("test.txt");
        assertTrue(file.exists());
    }
}
```

Listing 2

```
public class GlobalTimeout {

    @Rule //timeout nach 20 ms
    public Timeout timeout = new Timeout(20);

    @Test
    public void firstTest() {
        while (true) {}
    }

    @Test
    public void secondTest() {
        for (;;) {}
    }
}
```

Listing 3

```
public class ExpectedExceptionWithRule {

    int[ ] threeNumbers = { 1, 2, 3 };

    @Rule public ExpectedException thrown =
            ExpectedException.none();

    @Test
    public void exception() {
        thrown.expect(ArrayIndexOutOfBoundsException.class);
        threeNumbers[3] = 4;
    }

    @Test
    public void exceptionWithMessage() {
        thrown.expect(ArrayIndexOutOfBoundsException.class);
        thrown.expectMessage("3");
        threeNumbers[3] = 4;
    }
}
```

Listing 4

lich einem Aspekt in der aspektorientierten Programmierung (AOP) kann die Rule Code vor, nach oder anstelle einer Testmethode ausführen [2]. Hinter dieser abstrakten Beschreibung steckt ein mächtiges Werkzeug, wie die folgenden Beispiele zeigen. JUnit selbst liefert fünf Rules mit, an denen wir den praktischen Einsatz zeigen (der Quellcode aller Beispiele ist auf GitHub verfügbar [3]).

Temporäre Dateien

Beim Testen von Code, der Dateioperationen ausführt, steht man häufig vor dem Problem, dass der Test temporär eine Datei benötigt, die nach dem Test wieder gelöscht werden soll. Bisher brachte man den entsprechenden Code in @Before- und @After-Methoden unter, wie Listing 1 zeigt.

Dieser Test kann mit der „Temporary Folder"-Rule wesentlich kürzer und prägnanter formuliert werden, da die Rule den Framework-Code kapselt. Um die Rule zu verwenden, muss innerhalb des Tests ein Feld vom Typ „TemporaryFolder" angelegt werden. Dieses Feld muss „public" sein und mit der Annotation „@Rule" markiert werden, sodass JUnit die Rule erkennt. So markierte Rules wirken sich auf die Ausführung aller Testmethoden einer Testklasse aus (siehe Listing 2).

Die Testmethode „test()" legt mithilfe der „TemporaryFolder"-Rule die Datei „test.txt" an und überprüft danach, ob die Datei erzeugt wurde. Doch wo wurde sie erzeugt? Der Name „TemporaryFolder" suggeriert es bereits: in einem temporären Ordner. Doch die Rule legt die Datei nicht nur an, sondern löscht sie nach dem Test auch wieder, inklusive des temporären Ordners.

Timeout

Es kommt gelegentlich vor, dass man Code schreibt, der versehentlich Endlosschleifen enthält. Ein JUnit-Test, der diese Codestellen testet, läuft in diese Endlosschleifen. Bei Verwendung der „Timeout"-Rule schlagen solche Tests fehl, da sie nicht innerhalb der vorgegebenen Zeit beendet werden (siehe Listing 3).

Führt man diesen Test aus, schlagen beide Testmethoden fehl. Würde man die Rule nicht verwenden, liefe dieser Test

```
public class ErrorCollectingTest {

    @Rule
    public ErrorCollector collector = new ErrorCollector();

    @Test
    public void test() {
        collector.checkThat(1 + 1, is(3));
        collector.addError(new Exception("sth went wrong"));
    }
}
```

Listing 5

```
public class NameRuleTest {
    @Rule
    public TestName test = new TestName();

    @Test
    public void test() {
        assertThat(test.getMethodName(), is("test"));
    }
}
```

Listing 6

```
public class ProvideSystemProperty extends ExternalResource {

    private final String key, value;
    private String oldValue;

    public ProvideSystemProperty(String key, String value) {
        this.key = key;
        this.value = value;
    }

    @Override
    protected void before() {
        oldValue = System.getProperty(key);
        System.setProperty(key, value);
    }

    @Override
    protected void after() {
        if (oldValue == null) {
            System.clearProperty(key);
        } else {
            System.setProperty(key, oldValue);
        }
    }
}
```

Listing 7

endlos. Wer bisher den „timeout"-Parameter der „@Test"-Annotation verwendet hat, kann diesen durch die „Timeout"-Rule ersetzen. Die Rule bietet den Vorteil, dass sie nur einmal in der Klasse definiert werden muss und dann für alle Testmethoden gilt.

Erwartete Exceptions

Schon bisher kann das Auftreten von Exceptions mit dem „expected"-Parameter der „@Test"-Annotation getestet werden. Die „ExpectedException"-Rule erweitert die Test-Möglichkeiten für Exceptions. Damit lassen sich neben der Klasse auch die Message und mittels Hamcrest-Matchern sogar beliebige Details der geworfenen Exception testen (siehe Listing 4).

Fehler sammeln

Üblicherweise bricht ein Test nach der ersten fehlgeschlagenen Assertion ab. Will man in einem Test trotzdem alle Assertions abarbeiten, kann man den „ErrorCollector" verwenden. Er sammelt fehlgeschlagene Assertions innerhalb einer Testmethode und gibt am Ende eine Liste der Fehlschläge aus. So kann man etwa alle Elemente in einer Liste überprüfen und den Test erst am Ende fehlschlagen lassen, wenn die Überprüfung eines oder mehrerer Elemente missglückt ist (siehe Listing 5). Wenn man diesen Test ausführt, erhält man zwei Fehlernachrichten mit jeweils einem Stacktrace, der einen zu der Zeile im Programmcode führt, an der die Überprüfung fehlgeschlagen ist.

Testname

Um innerhalb einer Testmethode auf deren Namen zuzugreifen, kann man die „Test Name"-Rule verwendet (siehe Listing 6).

Die von JUnit bereitgestellten Rules sind nur der Anfang. Wer sich das Schreiben von Tests erleichtern will, kann seine eigenen Rules schreiben. Das sind letztendlich Klassen, die das Interface „TestRule" mit der Methode „apply(...)" implementieren. Für die häufigsten Anwendungsfälle greift uns JUnit unter die Arme und stellt die drei Template-Klassen „ExternalResource", „TestWatcher" und „Verifier" zur Verfügung.

Bereitstellung externer Ressourcen

Vielfach werden, insbesondere bei Integrationstests, externe Ressourcen wie Dateien,

Server oder Verbindungen benötigt. Diese müssen dem Test zur Verfügung gestellt und nach dessen Ausführung wieder aufgeräumt werden. Dieses Ressourcenhandling lässt sich recht einfach mit einer Rule abbilden, indem man von der Basisklasse „ExternalResource“ ableitet. In der neuen Rule überschreibt man die „before()“-Methode, um die Ressource bereitzustellen, und die „after()“-Methode, um sie nach dem Test wieder aufzuräumen. Ein Beispiel hierfür ist die „TemporaryFolder“-Rule, die in der „before()“-Methode ein neues Verzeichnis erstellt und es in der „after()“-Methode wieder löscht.

Wie einfach sich eine solche Rule schreiben lässt, demonstriert das folgende Beispiel. Möchte man für einen Test sicherstellen, dass eine System-Property einen bestimmten Wert hat und nach dem Test der alte Wert wiederhergestellt wird, könnte man die Methoden „before()“ und „after()“ wie in Listing 7 implementieren. Schon kann man die Rule in einem Test verwenden (siehe Listing 8).

Da man mit einer Rule Code vor und nach dem Aufruf der Testmethoden ausführen kann, lässt sich damit eine Benachrichtigung über die Testausführung realisieren. Dazu stellt JUnit die abstrakte Oberklasse „TestWatcher“ bereit. Diese besitzt vier leer implementierte Methoden, die man nach Bedarf überschreiben kann (siehe Listing 9). Die Benutzung in einem Test sieht dann so wie in Listing 10 aus.

Überprüfungen nach den Tests

Das dritte von JUnit zur Verfügung gestellte Template ist der „Verifier“. Dort kann man die Methode „verify()“ überschreiben, die nach jedem erfolgreichen Test ausgeführt wird. In dieser Methode lassen sich zusätzliche Überprüfungen unterbringen, die im Fehlerfall eine Exception werfen, um den Test doch noch scheitern zu lassen. Eine Beispiel-Implementierung von „Verifier“ ist der zuvor vorgestellte „ErrorCollector“. Während des Testlaufs sammelt er alle fehlgeschlagenen Assertions und wirft im Fehlerfall eine „MultipleFailureException“ am Ende des Tests.

TestRule implementieren

Anstatt eines der Templates zu verwenden, kann man das Interface „TestRule“

```
public class SomeTestUsingSystemProperty {

    @Rule
    public ProvideSystemProperty property =
        new ProvideSystemProperty("someKey","someValue");

    @Test
    public void test() {
        assertThat(System.getProperty("someKey"),
                is("someValue"));
    }
}
```

Listing 8

```
starting(), succeeded(), failed() und finished():
public class BeepOnFailure extends TestWatcher {

    @Override
    protected void failed(Throwable e, Description
desc) {
        Toolkit.getDefaultToolkit().beep();
    }
}
```

Listing 9

```
public class FailingTestThatBeeps {

    @Rule
    public BeepOnFailure beep = new BeepOnFailure();

    @Test
    public void test() {
        fail();
    }
}
```

Listing 10

```
public Statement apply(Statement base, Description desc) {
    return new Statement() {
        @Override
        public void evaluate() throws Throwable {
            before();
            try {
                base.evaluate();
            } finally {
                after();
            }
        }
    };
}
```

Listing 11

```
@RunWith(Suite.class)
@SuiteClasses({A.class, B.class, C.class})
public class UsesExternalResource {
   public static Server myServer = new Server();

   @ClassRule
   public static TestRule connect = new ExternalResource() {

      @Override protected void before() throws Throwable {
         myServer.connect();
      };

      @Override protected void after() {
         myServer.disconnect();
      };
   };
}
```

Listing 12

```
public class CombiningMultipleRules {

   @Rule public TestRule beep = new BeepOnFailure();
   @Rule public ExpectedException thrown =
                              ExpectedException.none();
   @Rule public TestName test = new TestName();

   @Test
   public void test() throws Exception {
      thrown.expect(IllegalArgumentException.class);
      throw new Exception("Hello from " +
                        test.getMethodName());
   }
}
```

Listing 13

```
public class UseRuleChain {
   @Rule
   public TestRule chain = RuleChain
         .outerRule(new LoggingRule(„outer rule"))
         .around(new LoggingRule(„middle rule"))
         .around(new LoggingRule(„inner rule"));
   @Test
   public void test() {}
}
```

Listing 14

auch direkt implementieren. Dieses Interface hat genau eine Methode: „Statement apply(Statement base, Description description);".Das erste Argument „base" kapselt den auszuführenden Test, der sich mittels„evaluate()" ausführen lässt. Die„description" stellt Informationen zum Test zur Verfügung (wie den Testnamen). Der Rückgabewert der Methode ist ein „Statement", das anstelle des Tests ausgeführt wird. Üblicherweise delegiert das neue „Statement" den Aufruf von „evaluate()" an den ursprünglichen Test und führt zusätzlich weitere Methoden aus. Der folgende Code zeigt beispielhaft die leicht abgewandelte Implementierung des „ExternalResource"-Templates (siehe Listing 11). Hier wird zuerst die Template-Methode „before()" ausgeführt, dann der Test selbst mittels „base.evaluate()" und zum Schluss die zweite Template-Methode „after()".

Regeln auf Klassenebene

Alle Rules, die wir bisher gesehen haben, wurden für jede Methode einzeln angewandt, genauso wie Methoden, die mit „@Before" und „@After" annotiert sind, vor beziehungsweise nach jedem Test ausgeführt werden. Manchmal möchte man allerdings die Möglichkeit haben, Code nur einmal vor der ersten bzw. nach der letzten Testmethode in einer Klasse auszuführen. Ein häufiger Anwendungsfall sind Integrationstests, die eine Verbindung zu einem Server aufbauen und wieder schließen müssen. Das war bisher nur mit den Annotations „@BeforeClass" beziehungsweise „@AfterClass" möglich; Rules konnte man dazu nicht verwenden. Um dieses Problem zu lösen, wurde in JUnit 4.9 die „@ClassRule"-Annotation eingeführt.

Um eine „ClassRule" zu verwenden, annotiert man ein Feld in der Testklasse, das analog zu „@BeforeClass-/@AfterClass"-Methoden „public" und „static" sein muss. Der Typ des Felds muss wie bei der „@Rule"-Annotation das „TestRule"-Interface implementieren. Eine solche Rule lässt sich nicht nur in einer normalen Testklasse verwenden, sondern auch in einer Test-Suite [4], wie Listing 12 zeigt.

Mehrere Regeln kombinieren

Einen weiteren Vorteil von Rules gegenüber Hilfsmethoden in Test-Oberklassen stellt ihre Kombinierbarkeit dar. Es lassen sich beliebig viele Rules in einem Test verwenden (siehe Listing 13).

Das funktioniert wunderbar, solange die Rules voneinander unabhängig sind. JUnit macht absichtlich keinerlei Zusicherungen, was die Reihenfolge der Abarbeitung von Rules angeht [5]. Manchmal möchte man aber dennoch eine bestimmte Reihenfolge vorgeben. Angenommen, man hat zwei Rules, von denen die erste eine bestimmte Ressource zur Verfügung stellt, die von der zweiten Rule benutzt wird. Dann möchte man sehr wohl sicher-

stellen, dass zuerst die Ressource bereitgestellt wird, bevor sie konsumiert wird. Dafür wurde in JUnit 4.10 die „RuleChain"-Klasse eingeführt. „RuleChain" implementiert selbst das „TestRule"-Interface, kann also verwendet werden wie eine normale Rule [6] (siehe Listing 14).

Wenn man diesen Test ausführt, erhält man folgende Ausgabe:

```
starting outer rule
starting middle rule
starting inner rule
finished inner rule
finished middle rule
finished outer rule
```

Die erste Regel (outer rule) umschließt also die mittlere (middle rule) und diese wiederum die dritte und letzte (inner rule).

Schreib deine eigenen Regeln!

Warum sollte man Rules verwenden? Ein großer Pluspunkt von Rules ist ihre Wiederverwendbarkeit. Sie ermöglichen, häufig benutzten Code, der bisher in „@Before/@After"-Methoden oder einer Testoberklasse stand, in eine eigene „TestRule"-Klasse auszulagern, die nur eine Verantwortlichkeit hat.

Ein weiterer Vorteil ist die Kombinierbarkeit von Rules. Wie wir in diesem Artikel gesehen haben, lassen sich beliebig viele Regeln in einem Test verwenden, sowohl auf Klassen- als auch auf Methodenebene. Viele Dinge, für die es in der Vergangenheit eines eigenen Test Runners bedurfte, lassen sich jetzt mit Rules implementieren. Da man immer nur einen Test Runner, aber beliebig viele Rules verwenden kann, stehen einem deutlich mehr Möglichkeiten offen.

Rules sind die Umsetzung von Delegation statt Vererbung für Unit-Tests. Wo früher Testklassenhierarchien mit Utility-Methoden gewuchert sind, kann man jetzt auf einfache Art und Weise verschiedene Rules kombinieren.

Fazit

Die vorgestellten, konkreten Rules demonstrieren lediglich die Vielfältigkeit der Einsatzmöglichkeiten. Eigene Regeln zu schreiben ist Dank der zur Verfügung gestellten Template-Klassen einfach. Erst diese Erweiterbarkeit macht Rules zu einem wirklichen Novum.
Die Macher von JUnit setzen jedenfalls für die Zukunft von JUnit voll auf den Einsatz und die Erweiterung von Rules. Kent Beck schreibt darüber in seinem Blog [7]: „Maybe once every five years unsuspectedly powerful abstractions drop out of a program with no apparent effort."

Links und Literatur

1. Kent Beck, Simple Smalltalk Testing With Patterns: http://www.xprogramming.com/testfram.htm
2. Blog von Jens Schauder: http://blog.schauderhaft.de/2009/10/04/junit-rules/
3. Source-Code der Beispiele auf GitHub: http://marcphilipp.github.com/junit-rules/
4. JUnit 4.9 Release Notes: http://github.com/KentBeck/junit/blob/master/doc/ReleaseNotes4.9.txt
5. Mailing List Post von Kent Beck über das Design von Rules: http://tech.groups.yahoo.com/group/junit/message/23537
6. JUnit 4.10 Release Notes: http://github.com/KentBeck/junit/blob/master/doc/ReleaseNotes4.10.txt
7. Blog von Kent Beck: http://www.threeriversinstitute.org/blog/?p=155

Marc Philipp
marc@andrena.de

Stefan Birkner
mail@stefan-birkner.de

Marc Philipp studierte Informatik an der Universität Karlsruhe (TH) und ist seit 2008 bei der andrena objects ag als Software-Entwickler und Entwickler-Coach tätig. Neben seiner täglichen Arbeit beschäftigt er sich mit Entwicklungswerkzeugen wie JUnit und Usus (projectusus.org).

Stefan Birkner arbeitet bei der Immobilien Scout GmbH. Dort ist er mitverantwortlich für die Weiterentwicklung der Suche. Neben dieser Tätigkeit beschäftigt er sich mit dem Testen von Software und beteiligt sich an der Entwicklung von JUnit und des Maven-Surefire-Plugins.

Vorschau

Java aktuell – Sommer 2012

Die nächste Ausgabe erscheint am 2. Mai 2012

Falls Sie einen Artikel in der nächsten Java aktuell veröffentlichen möchten, schicken Sie bitte vorab Ihren Themenvorschlag an redaktion@ijug.eu, Redaktionsschluss ist am 1. März 2012

Weaving, Instrumentation, Enhancement: Was ein JPA-Provider so alles macht

Marc Steffens und Bernd Müller, Ostfalia - Hochschule für angewandte Wissenschaften

Ein JPA-Provider muss die in der Spezifikation beschriebene Semantik von Assoziationen gewährleisten. Dazu müssen beispielsweise aus einer einfachen Return-Anweisung eines Attributs eine komplexe SQL-Anweisung erzeugt, diese ausgeführt und das Ergebnis aufbereitet werden. Um dies zu unterstützen, stellt JPA im Interface „ClassTransformer" Möglichkeiten bereit, eine Entity-Klasse auf Byte-Code-Ebene zu transformieren, die überarbeitete Version in die JVM zu laden und der Anwendung zur Verfügung zu stellen.

Der Artikel zeigt allgemein die Aufgabenstellung eines JPA-Providers bezüglich Assoziationen und die Realisierung durch die drei populären JPA-Provider EclipseLink, Hibernate und OpenJPA. Da die jeweiligen Realisierungen relativ aufwändig und komplex sind, geben wir diese nur im Ansatz wieder, versuchen aber trotzdem, dem Leser einen Eindruck der von den JPA-Providern geleisteten Aufgaben zu geben.

Zur Beschreibung von Assoziationen existieren in JPA die vier Annotationen „@OneToOne", „@OneToMany", „@ManyToOne" und „@ManyToMany". Mit JPA 2.0 kamen Möglichkeiten für Element-Collections hinzu. Um den Umfang nicht zu sprengen, konzentrieren wir uns exemplarisch auf 1:1-Beziehungen. Als Beispiel dient ein Ausschnitt aus dem Hochschulalltag: Zwischen den Entity-Klassen „Student" und „Studentenausweis" besteht eine 1:1-Beziehung, die, wie im folgenden Beispiel dargestellt, realisiert werden kann (siehe Listing 1).

```
public class Student {
  ...
  @OneToOne
  Studentenausweis ausweis;
  ...
  public Studentenausweis getAusweis() {
    return ausweis;
  }
  ...
}
```

Listing 1

JPA sieht für Objekt-wertige Assoziationsziele das frühe Laden (eager loading), für Collection-wertige Assoziationsziele das späte Laden (lazy loading) als Default für die Implementierung der Assoziation vor. Beim späten Laden einer Assoziation wird der JPA-Provider in der Regel eine Select-Anweisung an das Datenbank-System absetzen, um das Assoziationsziel in die JVM zu laden. Statt der einfachen Return-Anweisung im Quell-Code muss also eine SQL-Anweisung erzeugt und ausgeführt sowie das Ergebnis in entsprechende Java-Objekte eingepackt werden. Um dies zu unterstützen, sieht JPA das Interface „ClassTransformer" im Package „javax.persistence.spi" vor. Die einzige Methode „transform()" dieses Interface repliziert die Methode desselben Namens im Interface „ClassFileTransformer" im Java-SE Package „java.lang.instrument" (siehe Listing 2).

Die drei populären JPA-Provider EclipseLink, Hibernate und OpenJPA verwenden diese und andere Möglichkeiten, um Transformationen am Byte-Code von Entity-Klassen vorzunehmen, benutzen aber verschiedene Bezeichnungen, um diese Transformationen zu beschreiben. EclipseLink nennt dies „Weaving", Hibernate „Instrumentation" und OpenJPA „Enhancement".

Der Artikel soll einen Eindruck davon vermitteln, was ein JPA-Provider tun muss, um seiner Aufgabe gerecht zu werden. Da die Aufgaben relativ umfangreich sind, werden nur die ersten Schritte beschrieben; es wird nicht bis zur SQL-Generierung vorgedrungen. Bei Interesse am generierten SQL raten die Autoren, den Log-Level der drei Provider entsprechend zu erhöhen und das Log zu studieren.

Wir schauen uns im Folgenden die jeweiligen Vorgehensweisen zur Veränderung des Byte-Codes an. Um das Beispiel möglichst einfach zu halten, beschränken wir uns auf die 1:1-Beziehung zwischen „Student" und „Studentenausweis" und hier wiederum auf den Getter. Da eine solche Beziehung im Default früh geladen wird, spätes Laden aber aufwändigere und für unsere Darstellung geeignetere Mechanismen benötigt, definieren wir explizit das späte Laden: „@OneToOne(fetch

```
byte[ ] transform(java.lang.ClassLoader loader,
        java.lang.String className,
        java.lang.Class<?> classBeingRedefined,
        java.security.ProtectionDomain protectionDomain,
        byte[ ] classfileBuffer)
        throws
    java.lang.instrument.IllegalClassFormatException
```

Listing 2

```
<target name="weaving" description="weaving">
  <weave source="build/project.jar" target="woven.jar"
     persistenceinfo="../src" loglevel="FINEST">
    <classpath>
      <fileset dir="${lib.dir}" includes="**/*.jar" />
    </classpath>
  </weave>
</target>
```

Listing 3

```
public Studentenausweis getAusweis() {
    return _persistence_get_ausweis();
}

public Studentenausweis _persistence_get_ausweis() {
    _persistence_checkFetched(„ausweis");
    _persistence_initialize_ausweis_vh();
    this.ausweis = ((Studentenausweis)this._persistence_ausweis_
vh.getValue());
    return this.ausweis;
}
```

Listing 4

= FetchType.LAZY)". Die Realisierung des späten Ladens erfordert eine Byte-Code-Manipulation von EclipseLink und OpenJPA, während Hibernate auch ohne eine solche Transformation spät laden kann.

EclipseLink: Weaving

EclipseLink nennt die Byte-Code-Transformation „Weaving" und unterscheidet zwischen der statischen Transformation nach dem Kompilieren mithilfe einer Ant-Task beziehungsweise der Kommandozeile und der dynamischen Transformation zum Ladezeitpunkt mithilfe eines Java-Agenten. Als Werkzeug wird die Bibliothek „ASM" [1] verwendet.

Zur statischen Transformation bietet EclipseLink eine Ant-Task mit Namen „StaticWeaveAntTask" an, die hier „weave" heißt und die in den Attributen „source" und „target" das Quell- und Ziel-Jar der Transformation erwartet (siehe Listing 3).

Die dynamische Variante wird über einen Java-Agenten, der sich im Jar der EclipseLink-Implementierung befindet, realisiert.

```
java -javaagent:eclipselink.jar MainClass
```

Weitere Details sowohl zum statischen als auch zum dynamischen Weaving findet man in [2].

Der folgende Code zeigt das Ergebnis der Klassentransformation für die Methode „getAusweis()" (siehe Listing 4).

Um das späte Laden von Attributen zu ermöglichen, verwendet EclipseLink einen sogenannten „Value-Holder". Das Interface „ValueHolderInterface" im Package „org.eclipse.persistence.indirection" definiert Methoden zum Lesen und Schreiben von „Object"-Instanzen. Das oben verwendete Attribut „persistence_ausweis_vh" ist wie in Listing 5 deklariert. Das verwendete Interface ist ein Sub-Interface von „ValueHolderInterface". Beim Zugriff auf das Attribut wird geprüft, ob bereits ein Laden aus der Datenbank stattgefunden hat. Falls nicht, wird dies nachgeholt.

```
protected WeavedAttributeValueHolderIn-
terface
    _persistence_ausweis_vh;
```

Listing 5

Hibernate: Instrumentation

Hibernate nennt die Klassentransformation „Instrumentation" und stellt ebenfalls eine Ant-Task hierfür bereit. Intern wird die Bibliothek „Javassist" [3] verwendet, ebenfalls ein JBoss-Projekt. Das folgende Listing zeigt die Verwendung dieser Ant-Task. Eine alternative Realisierung mit einem Java-Agenten existiert bei Hibernate nicht (siehe Listing 6). Bemerkung: Hibernate unterstützt das späte Laden von 1:1-Beziehungen auch ohne explizite Instrumentierung. Diese bietet jedoch bestimmte Optimierungen und wird zum Beispiel für das korrekte Verhalten von Fetch-Groups zwingend benötigt.

Die „InstrumentTask" erweitert die Klassen um einen sogenannten „Field-Handler" (Interface „FieldHandler"). Das Lesen und Schreiben der Attribute wird nun über diesen „FieldHandler" realisiert. Listing 7 zeigt die Modifikation des Beispiel-Getters durch die „InstrumentTask".

Ein „FieldHandler" stellt für die primitiven Datentypen Lese- und Schreibmethoden bereit: „readInt()", „writeInt()", „readDouble()", „writeDouble()" etc. Die entsprechende Lesemethode für Objekte „readObject()" gibt Listing 8 wieder. Man erkennt hier, dass die Arbeit auf ein Proxy verlagert wird, das letztendlich für den SQL-Zugriff verantwortlich ist.

OpenJPA: Enhancement

Die Transformation von Entity-Klassen wird von OpenJPA „Enhancement" genannt. OpenJPA unterstützt sowohl die Transfor-

```
<target name="instrument">
        <taskdef name="instrument" classname=
„org.hibernate.tool.instrument.javassist.InstrumentTask">
                <classpath refid="classpath"/>
        </taskdef>
        <instrument verbose="true">
          <fileset dir="../build/classes">
            <include name="**/*.class"/>
          </fileset>
        </instrument>
</target>
```

Listing 6

```
public Studentenausweis getAusweis() {
      return $javassist_read_ausweis();
}
public Studentenausweis $javassist_read_ausweis(){
   if (getFieldHandler() == null) {
     return this.ausweis;
   }
   return (Studentenausweis) this.getFieldHandler
         .readObject(this, „ausweis", ausweis);
}
```

Listing 7

```
public Object readObject(Object target, String name,
                  Object oldValue) {
  Object value = intercept( target, name, oldValue );
  if (value instanceof HibernateProxy) {
    LazyInitializer li =
     ((HibernateProxy)value).getHibernateLazyInitializer();
    if ( li.isUnwrap() ) {
      value = li.getImplementation();
    }
  }
  return value;
}
```

Listing 8

```
  <target name="enhance">
    <path id="jpa.classpath">
      <pathelement location="../build/classes"/>
        <fileset dir="../lib">
          <include name="**/*.jar"/>
        </fileset>
    </path>
    <taskdef name="enhancer"
          classname="org.apache.openjpa.ant.PCEnhancerTask">
      <classpath refid="jpa.classpath"/>
    </taskdef>
    <enhancer>
      <classpath refid="jpa.classpath"/>
    </enhancer>
  </target>
```

Listing 9

```
public Studentenausweis getAusweis() {
   if (this.pcStateManager == null) {
         return pcgetAusweis();
   }
   int i = pcInheritedFieldCount + 0;
   this.pcStateManager.accessingField(i);
   return pcgetAusweis();
}
```

Listing 10

mation mit einer Ant-Task als auch die über einen Java-Agenten. Als Realisierungshilfe wird „Serp" [5] verwendet. Der folgende Ausschnitt eines Ant-Build-Files gibt die Struktur zur Verwendung der Task wieder.

Für das späte Laden von 1:1-Beziehungen muss das Enhancement erfolgen (siehe Listing 9).

Eine vollständige Übersicht über die Möglichkeiten des Enhancement von Entity-Klassen findet man in [6] und [7]. Das zentrale Interface von OpenJPA zur Zustandsverwaltung ist der „StateManager" im Package „org.apache.openjpa.enhance". Die überarbeitete Version des Beispiel-Getters zeigt der folgende Code-Ausschnitt (siehe Listing 10).

Wird auf den Getter der Entity-Klasse zugegriffen, so sorgt der „StateManager" durch die Methode „accessingField()" für das Laden des Attributs. Die auf den ersten Blick merkwürdige Berechnung des Wertes „i" durch Addition von 0 ist der Implementierung von OpenJPA geschuldet. Die Variable „i" repräsentiert einen Index in einem Array, in dem Eigenschaften wie Namen und Typen der Attribute hinterlegt sind. Die addierte Konstante (hier 0) wird zum Zeitpunkt des Enhancement berechnet und beschreibt den Index für dieses Attribut innerhalb des Arrays.

Gegenüberstellung

Tabelle 1 stellt die Möglichkeiten der statischen und dynamischen Byte-Code-Manipulation der drei JPA-Provider gegenüber. Außerdem sind die zur Byte-Code-Manipulation verwendeten Werkzeuge beziehungsweise Bibliotheken angegeben.

Fazit

Wir haben in diesem Artikel motiviert, dass JPA-Provider zur korrekten Imple-

	EclipseLink	Hibernate	OpenJPA
Byte-Code-Manipulation	Ant / Java-Agent	Ant	Ant / Java-Agent
Bibliothek	ASM	Javassist	Serp

Tabelle 1: Gegenüberstellung der JPA-Provider

mentierung der JPA-Spezifikation eine Reihe von Änderungen an Entity-Klassen vornehmen müssen. Diese Änderungen sollten sinnvollerweise nicht auf Quell-Code-Ebene, sondern im Byte-Code erfolgen. JPA sieht hierzu das Interface „ClassTransformer" vor. Um die eigentlichen Code-Transformationen vorzunehmen, verwenden die Provider verschiedene Werkzeuge, namentlich ASM, Javassist und Serp. Am Beispiel einer 1:1-Beziehung haben wir uns die oberste Ebene der entsprechenden Transformationen näher angeschaut und durchaus unterschiedliche Ansätze bei EclipseLink, Hibernate und OpenJPA erkennen können.

Weitere Informationen

[1] http://asm.ow2.org/
[2] http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_%28ELUG%29#What_You_May_Need_to_Know_About_Weaving_JPA_Entities
[3] http://www.jboss.org/javassist
[4] http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/performance.html
[5] http://serp.sourceforge.net/
[6] http://openjpa.apache.org/enhancement-with-ant.html
[7] http://openjpa.apache.org/docs/latest/ref_guide_pc_enhance.html#ref_guide_pc_enhance_unenhanced_types

Marc Steffens
m-th.steffens@ostfalia.de

Bernd Müller
bernd.mueller@ostfalia.de

Marc Steffens studiert IT-Management an der Ostfalia und beschäftigt sich in seiner Bachelor-Arbeit unter anderem mit den Möglichkeiten, die Java-Bytecode-Manipulationen bieten.

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor mehrerer Bücher zu den Themen JSF, JPA und JBoss Seam.

Die iJUG-Mitglieder auf einen Blick

iJUG Verbund

Java User Group Deutschland e.V.
http://www.java.de

DOAG Deutsche ORACLE Anwendergruppe e. V.
http://www.doag.org

Java User Group Stuttgart e.V. (JUGS)
http://www.jugs.de

Java User Group Köln
http://www.jugcologne.eu

Java User Group München (JUGM)
http://www.jugm.de

Java User Group Metropolregion Nürnberg
http://www.source-knights.com

Java User Group Ostfalen
http://www.jug-ostfalen.de

Java User Group Saxony
http://www.jugsaxony.org

Sun User Group Deutschland e.V.
http://www.sugd.de

Swiss Oracle User Group (SOUG)
http://www.soug.ch

Der iJUG möchte alle Java-Usergroups unter einem Dach zu vereinen. So können sich alle interessierten Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne beim iJUG melden unter: office@ijug.eu

Das Eclipse-Modeling-Framework

Jonas Helming und Maximilian Kögel, EclipseSource München GmbH

Mit dem Eclipse-Modeling-Framework (EMF) können Entitäten einer Anwendung modelliert und als Java-Klassen generiert werden. Dies ermöglicht einen sehr kurzen Entwicklungszyklus vom Datenmodell zu einer lauffähigen ersten Anwendung, in der Entitäten, Attribute und Referenzen bereits in einer Benutzeroberfläche sichtbar sind. Dieser Artikel führt in die Basisfunktionen von EMF ein und zeigt, wie generierte Klassen direkt in einem Beispieleditor getestet werden können. In den weiteren Artikeln stellen wir die API von EMF sowie Frameworks, die im Zusammenspiel mit EMF verwendet werden können, vor.

Buzz-Words in der Software-Welt haben ihren ganz eigenen Lebenszyklus, seien es nun „Agile Entwicklung", „Cloud" oder „Ontologien". Sie entwickeln sich von der Vision zum Hype, von der Universal-Lösung zu den ersten Rückschlägen. Wenn dann die Schaulustigen zum nächsten oder übernächsten Trend weitergezogen sind, einige Pilotprojekte gut oder schlecht gelaufen sind, bleibt das übrig, was wirklich sinnvoll ist. Dieser Rest entwickelt sich weiter und wird zum nicht mehr wegzudenkenden Teil der täglichen Entwicklungsarbeit.

Nicht anders verlief und verläuft es bei den Trends „Model-Driven Architecture" (MDA) und „Model-Driven Development" (MDD). Einer der Teile, die im Bereich MDA übriggeblieben sind, ist das Eclipse-Modeling-Framework (EMF). Es erlaubt, Modelle zu erstellen, aus denen strukturierte Datenmodelle, also Entitätsmodelle generiert werden können. Es geht somit um die Objekte der Anwendung, die im Wesentlichen Daten in Attributen und Referenzen zwischen den Objekten enthalten. Generiert werden Java-Klassen, die Getter- und Setter-Methoden enthalten, ganz ähnlich zu Java Beans. Die generierten Klassen enthalten aber zusätzlich bereits eine ganze Reihe von Funktionalitäten, wie beispielsweise einen Notifikationsmechanismus oder einen Beispieleditor.

EMF erlaubt einen sehr schnellen Durchstich bei der Entwicklung der Modelle. Es müssen lediglich die Entitäten des Datenmodells, deren Attribute und Referenzen modelliert werden. Das Ergebnis kann ohne weiteres Zutun bereits in einer laufenden Anwendung getestet werden. So kann sehr früh Feedback vom Kunden eingeholt werden, beispielsweise darüber, ob noch zusätzliche Attribute im Datenmodell benötigt werden.

EMF wurde vor mehr als zehn Jahren mit zwei Zielen entwickelt. Zum einen sollte die zeitintensive, manuelle Erstellung von Entitätsobjekten automatisiert werden. Zum anderen sollten diese in einem Standard erstellt sein. Das ermöglicht den Austausch von Daten zwischen verschiedenen Teilen einer Anwendung oder zwischen verschiedenen Anwendungen. IBM nutzte EMF in der Anfangszeit vor allem für Tools, beispielsweise um Konfigurationen zu persistieren. Um einem existierenden Standard zu folgen, sollte dazu zunächst der Standard Meta Object Facility (MOF) der Object Management Group (OMG) realisiert werden. Die Umsetzung zeigte, das MOF in der Praxis komplex und im Wesent-

lichen zu umfangreich war. Die endgültige Implementierung von EMF setzte schließlich nur eine Untermenge von MOF um, die wirklich benötigt wurde. Diese Untermenge wurde ihrerseits wieder zu einem Standard: essential Meta-Object-Facility oder eMOF. Diese Vorgeschichte führt zu einem der Hauptvorteile von EMF: Es ist ein pragmatischer Ansatz, der in der Praxis funktioniert.

Damit füllte EMF technologisch sehr früh eine Position im Eclipse-Ökosystem aus. Neun Jahre später ist es der De-facto-Standard für Entitätsmodellierung in Eclipse. Die Stabilität des Frameworks ermöglichte das Wachstum zahlreicher weiterer Frameworks um EMF herum. Diese visualisieren, persistieren oder transformieren EMF-Modelle. Dabei nutzen die meisten Frameworks die mitgenerierte Funktionalität der Modelle, beispielsweise um die Benutzeroberfläche bei Änderungen im Modell zu aktualisieren. Der Modellierungsbereich ist einer der aktivsten in der Eclipse-Welt. Wir nutzen den neunten Geburtstag von EMF, um eine Rundreise durch die Modellierungswelt anzutreten. In dieser und der nächsten Ausgabe werden die Grundlagen von EMF vorgestellt. Eine englische Version kann als Tutorial unter [4] heruntergeladen werden. In den kommenden Ausgaben beschreiben wir einige der zahlreichen Frameworks rund um EMF. Weitere Informationen zu EMF und dem Eclipse-Modeling-Projekt findet man unter [1].

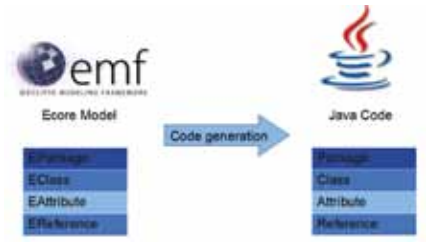


Abbildung 1: Codegenerierung in EMF

Abbildung 2: IDE vs. Runtime

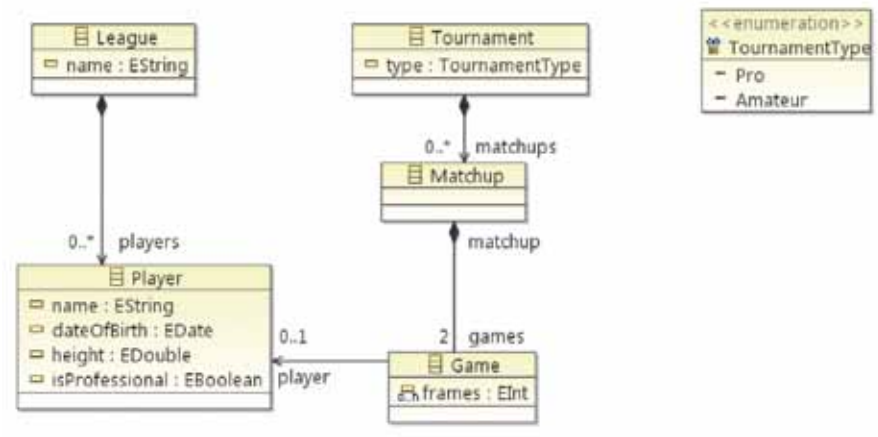


Abbildung 3: Das Beispielmodell

Grundlagen

Der erste Schritt in der Verwendung von EMF ist die Definition eines Modells. Das Modell definiert Entitäten, deren Attribute sowie Referenzen zwischen den Entitäten. Aus diesem Modell werden später Java-Klassen generiert. Das Standardformat für Modelle in EMF ist Ecore, die Umsetzung des bereits erwähnten eMOF. Alternativ können Modelle auch beispielsweise als UML-Klassendiagramm, als annotierte Java Interfaces oder in Rational Rose definiert werden; dieses Tutorial beschreibt mit Ecore den Standardweg. Um zwischen Modell und generiertem Code unterscheiden zu können, werden Konzepte in EMF mit dem Prefix „E" versehen, eine Klasse in Java wird aus einer EClass in EMF generiert (siehe Abbildung 1).

Um den generierten Code zu testen, kann aus der Eclipse IDE, die zum Modellieren verwendet wird, leicht eine zweite Eclipse-Instanz (Runtime) gestartet werden, die das kompilierte Modell enthält. In diesem Eclipse-Modell können dann Instanzen der generierten Java-Klassen erzeugt werden (siehe Abbildung 2).

Eine Randbemerkung zur Begrifflichkeit insbesondere mit Blick auf den manchmal verwirrenden Ausdruck „Meta-Modell". In diesem Artikel wird das Modell, in dem die Entitäten der Anwendung modelliert werden, mit „Modell", deren Instanzen mit „Modell-Instanz" bezeichnet. Das Ecore-Format, in dem das Modell ausgedrückt wird, ist damit das Meta-Modell. Bezeichnete man hingegen die Modell-Instanz als Modell, würde das Anwendungsmodell zum Meta-Modell, das Ecore zum Meta-Meta-Modell.

Das Beispielmodell

Dieser Artikel beschreibt zunächst die Grundlagen von EMF anhand eines Beispielmodells. Dieses bildet rudimentär den Aufbau einer Bowling-Liga ab. Der Fokus des Beispiels liegt dabei weniger auf der korrekten Abbildung der Bowling-Regeln

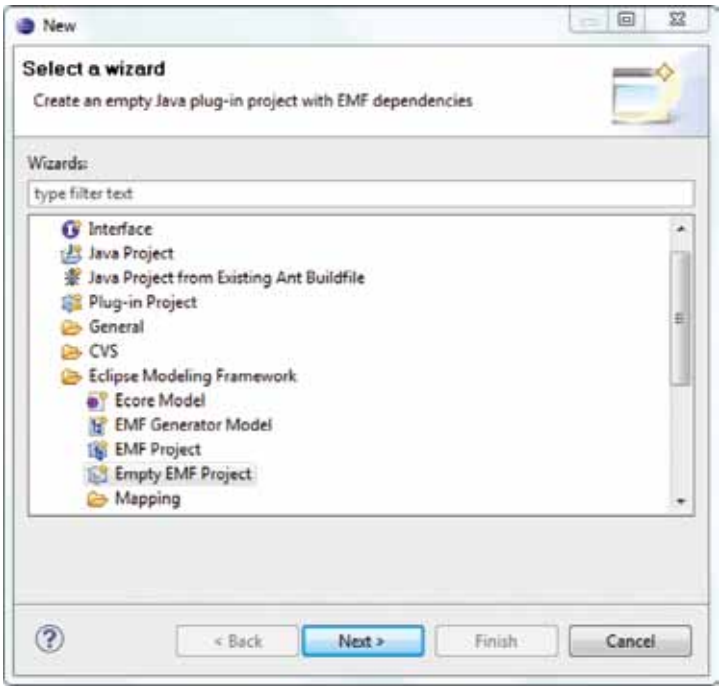


Abbildung 4: Erzeugen eines neuen EMF-Projekts

Abbildung 5: Erzeugen eines Ecore-Modells

als vielmehr auf einem Modell, das alle wichtigen Grundkonzepte von EMF enthält.

Abbildung 3 zeigt das Beispielmodell als UML-Diagramm. Eine „League" enthält beliebig viele „Player". „League" und „Player" enthalten jeweils Attribute wie einen „Namen" oder das „Geburtsdatum". Ein „Tournament" besteht aus beliebig vielen „Matchups", ein „Matchup" aus genau zwei „Games". Ein „Game" ist die „Serie" genau eines „Players" und enthält im Attribut „Frames" dessen Punkte. Das „Tournament" enthält schließlich eine Enumeration, die den Typ des Turniers festlegt.

Dieses Modell wird im EMF abgebildet, anschließend wird daraus Code generiert. Für den Einsatz von EMF steht eine vorbereitet Eclipse-Version, die Eclipse-Modeling-Tools, bereit, die schon die wichtigsten Komponenten enthält. Sie kann unter [2] heruntergeladen werden. Unter [3] steht die Beispiellösung dieses Tutorials. Diese kann zum Vergleichen verwendet werden.

Modellierung

Der erste Schritt der Modellierung besteht darin, ein neues, leeres EMF-Projekt im Workspace zu erzeugen (siehe Abbildung 4). Im Beispiel heißt es „org.eclipse.example.bowlingmodel".

Der zentrale Teil jedes Modellierungsprojekts ist das Modell, in unserem Fall im Ecore-Format definiert. Ein Ecore-Modell wird im Ordner „model" erstellt, im Beispiel heißt die Datei „bowling.ecore".

Nach dem Erzeugen des Ecore-Modells wird dieses im Standard-Ecore-Editor geöffnet. Damit kann das gesamte Modell in einer Baumansicht erzeugt und bearbeitet werden. Dies ist die grundlegendste Form, um ein Ecore-Modell zu erzeugen. Alternativ gibt es grafische und textuelle Editoren. Ein grafischer Editor der Ecore-Tools kann jederzeit über einen Rechtsklick auf das Modell und die Auswahl von „Initialize Ecore Diagram" aufgerufen werden. Wir beschränken uns jedoch auf den Standard-Editor. In diesem können Elemente wie Klassen oder Attribute über einen Rechtsklick erzeugt und beliebig per Drag and Drop verschoben werden. Die Eigenschaften einzelner Elemente lassen sich in der Properties View bearbeiten. Bevor mit der eigentlichen Modellierung begonnen wird, sollte das Root-Package des Modells eindeutig benannt sein. Dies identifiziert das Modell eindeutig, beispielsweise wenn in einer Anwendung mehrere Modelle gleichzeitig verwendet werden. Abbildung 6 zeigt den Namen und den Namespace des Beispielmodells.

Nun können als „Kinder" des Root-Package über einen Rechtsklick Klassen (EClasses) erzeugt werden. Abbildung 7 zeigt die erste Klasse des Beispielmodells. Der Name, in diesem Fall „Player", wird über

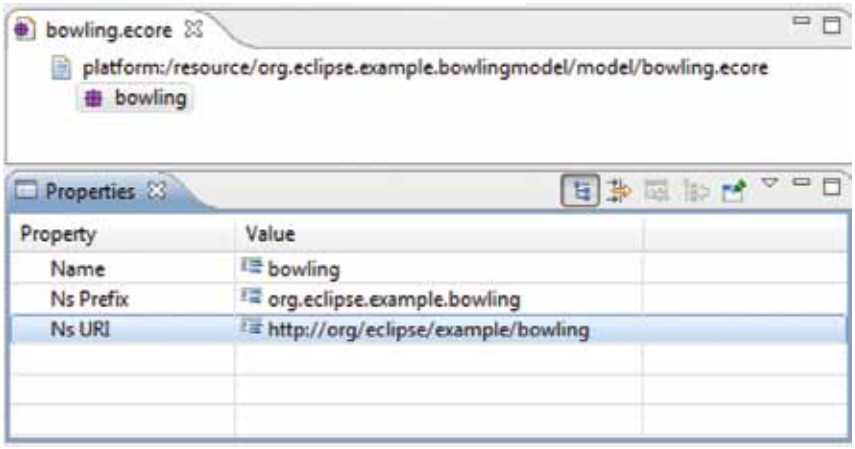


Abbildung 6: Benennung des Modells

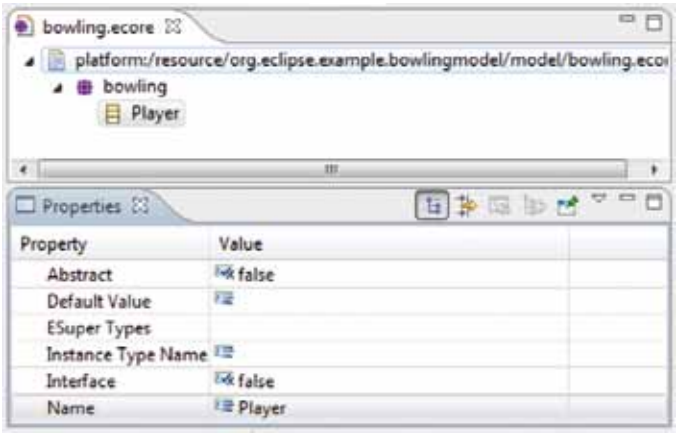


Abbildung 7: Anlegen der Klasse „Player"

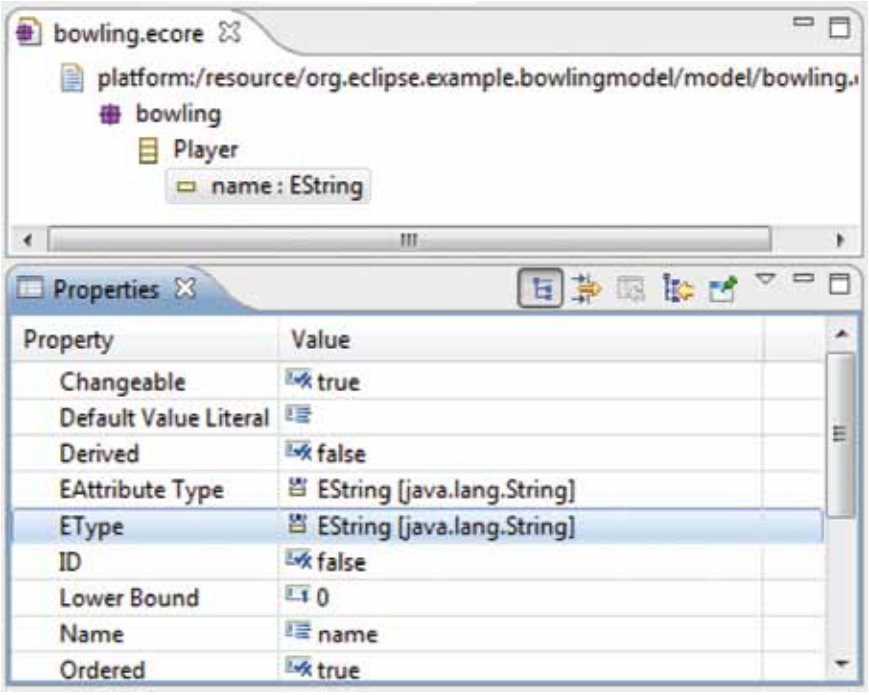


Abbildung 8: Anlegen eines Attributs

die Properties View angegeben. Klassennamen sind per Konvention wie auch in Java großgeschrieben.

Nun kann man durch einen Rechtsklick auf die Klasse Attribute (EAttribute) und Referenzen (EReference) erzeugen. In der Properties View können der Name eines Attributs und sein Typ bestimmt werden, im Beispiel ein Attribut „name" mit dem Typ „Estring" (siehe Abbildung 8). Attribute werden in EMF per Konvention kleingeschrieben, bei der Code-Generierung werden später Getter- und Setter-Methoden korrekt erzeugt, beispielsweise „getName()" und „setName()".

Alle einfachen Datentypen sind EMF und mit dem Prefix „E" versehen, beispielsweise „EDate". Auf die gleiche Weise können nun auch die übrigen Attribute der Klasse „Player" sowie die Klasse „League" erzeugt werden (vergleiche Abbildung 8).

Neben Klassen und einfachen Attributen bilden EMF-Modelle üblicherweise einen „Containment Tree". Das bedeutet, Elemente des Modells sind in anderen über eine Referenz enthalten (Containment). Das Modell bildet dadurch eine Baumstruktur, über die das Modell navigiert und serialisiert werden kann. Im Beispielmodell sind „Player" in der Klasse „League" enthalten. Diese Art der Referenz hat zwei Konsequenzen. Zum einen kann jeder Player nur in genau einer League zur gleichen Zeit referenziert sein. Zum anderen werden beim Löschen einer League gleichzeitig alle enthaltenen Player gelöscht.

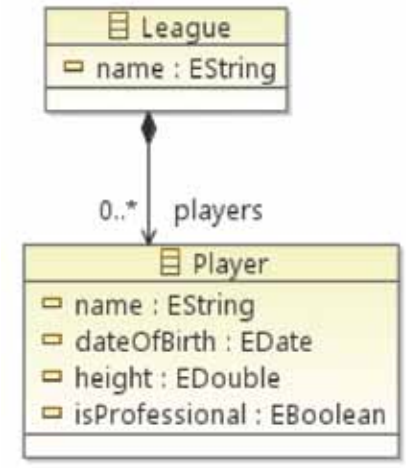


Abbildung 9: Die Containment-Referenz zwischen „League" und „Player"

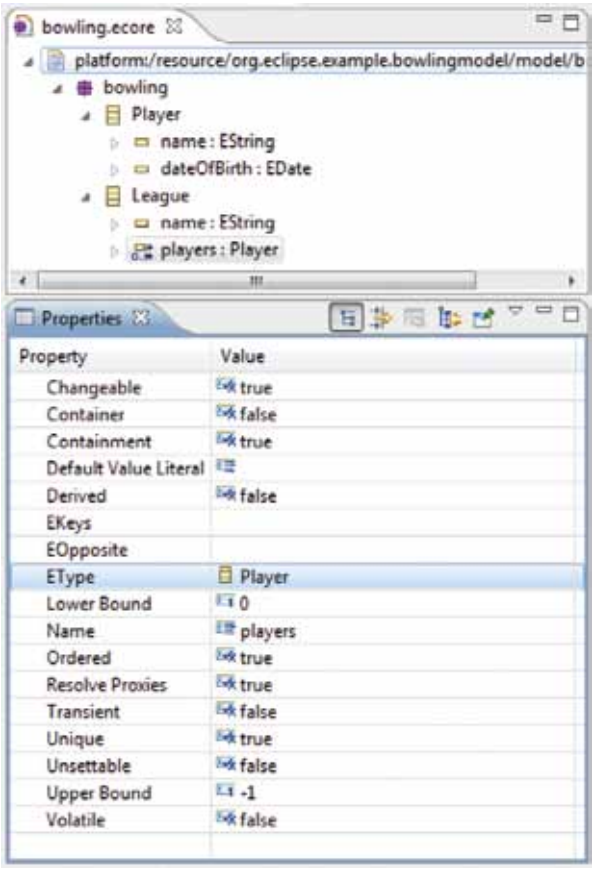


Abbildung 10: Eine Containment-Referenz in EMF

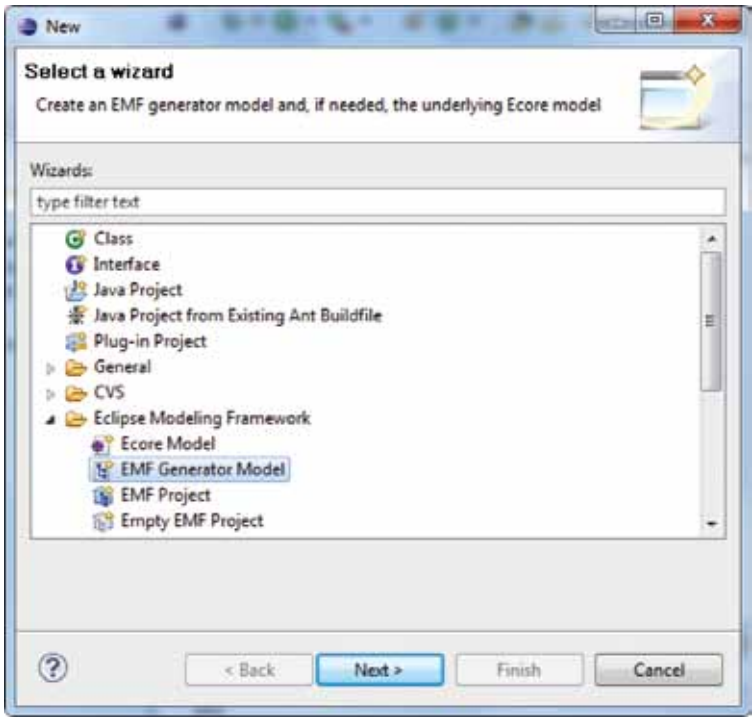


Abbildung 11: Anlegen eines Generator-Modells

Eine Referenz kann wie ein Attribut durch einen Rechtsklick auf eine Klasse erzeugt werden.

Im Beispiel zeigt die Referenz vom Element „League“ auf das Element „Player“. Daher wird sie durch einen Rechtsklick auf das Element „League“ erzeugt. Da die Referenz die „Player“ einer „League“ enthält, wird sie „players“ genannt. Setzt man das Property „Containment“ auf „true“, bildet die Referenz ein Containment. Ähnlich wie bei einem Attribut muss der Typ der Referenz angegeben werden. Der Typ ist die Klasse, auf den die Referenz zeigt, im Beispiel also „Player“. Weiterhin muss die Multiplizität der Referenz angegeben sein, im Beispiel kann eine League beliebig viele Player enthalten. Dies wird in EMF ausgedrückt, indem das Property „Upper Bound“ auf -1 gesetzt wird, das Äquivalent zu „beliebig viele“.

Code-Generierung

Aus diesem ersten Teil des Beispielmodells lässt sich nun bereits Code generieren. Die übrigen Teile des Modells können später hinzugefügt werden, worauf der Code erneut entsteht. Um Code zu generieren, ist zunächst ein Generator-Modell zu erzeugen.

Dies ist ein Konfigurationsmodell, das Einstellungsmöglichkeiten für die Code-Generierung enthält, beispielsweise dafür, wohin der Code generiert wird. „Modell“ deswegen, weil die Einstellungen für die Code-Generierung selbst ein EMF-Modell sind.

Ein Generator-Modell (bowling.genmodel) wird wie das Ecore-Modell selbst im Ordner „model“ erzeugt. Als Quelle wird das eben erstellte Ecore angegeben (Abbildung 11 und 12). Wird das Modell in einem anderen Format als Ecore definiert, können im Generator-Modell auch andere Quellen angegeben werden (siehe Abbildung 12).

Im Generator-Modell können zahlreiche Einstellungen vorgenommen werden, einige davon werden in den kommenden Teilen dieser Reihe vorgestellt. Für den ersten Wurf genügen die Standard-Einstellungen. Durch einen Rechtsklick auf den Root im Generator-Modell werden über die Auswahl von „Generate All“ vier Plug-ins generiert:

- *Modell*
  Das Modell-Plug-in enthält die generierten Entitäten (Java-Klassen), Packages, sowie Factory-Klassen, um Instanzen des Modells zu erzeugen. Es ist identisch mit dem bereits erzeugten Plug-in, in dem das Modell definiert wurde.
- *Edit*
  Das Edit-Plug-in enthält zusätzliche Klassen, um Instanzen des Modells in einer Anwendung anzuzeigen, beispielsweise Label Provider, die Icons für Elemente des Modells darstellen.
- *Editor*
  Das Editor-Plug-in ist ein Beispieleditor für das generierte Modell. Mit diesem können Instanzen des Modells erzeugt werden, das Modell kann also direkt ausprobiert werden.
- *Test*
  Das Test-Plug-in enthält Hüllenklassen für das Erstellen von Test Cases mit JUnit.

In der nächsten Ausgabe werden wir einen genaueren Blick auf den generierten Code sowie auf die API von EMF werfen. In diesem Teil sollen nun zunächst mit dem Beispiel-Editor Instanzen des erstellten Modells erzeugt werden. Dazu wird durch einen Rechtsklick auf ein beliebiges Plug-in und Auswahl von „Debug as => Eclipse Application" eine zweite Eclipse-Instanz gestartet, die die generierten Plug-ins enthält. Diese zweite Instanz enthält nun das kompilierte Modell sowie den generierten Beispiel-Editor. In dieser Instanz kann nun eine Modell-Instanz erzeugt werden. Dazu legt man ein leeres Projekt und darin ein Bowling Model „myLeague.bowling" an (siehe Abbildung 13 und 14).

Für die Modell-Instanz muss ein Root-Objekt festgelegt werden, dass alle anderen Elemente der Modell-Instanz enthält. Im Beispiel ist dies das Root-Objekt „League", das beliebig viele Player enthalten kann.

Nach dem Anlegen der Modell-Instanz öffnet sich der zuvor generierte Beispiel-Editor. Dieser funktioniert analog zum vorher verwendeten Ecore-Editor. Über einen Rechtsklick auf die „League" werden „Player" angelegt. Über die Properties View können die Attribute der angelegten

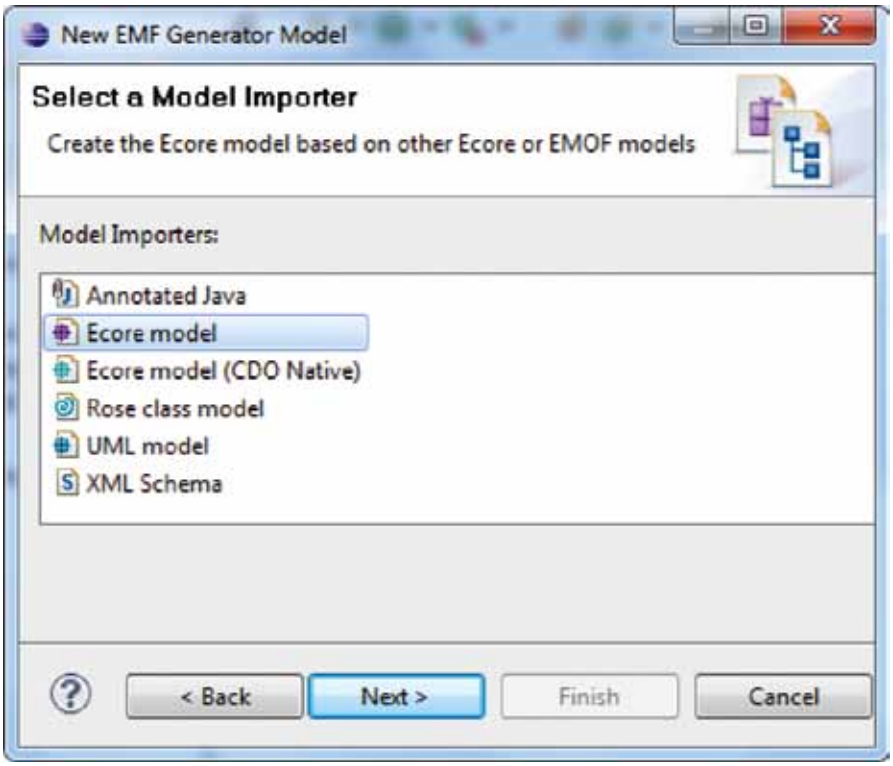


Abbildung 12: Angeben der Quelle für die Generierung

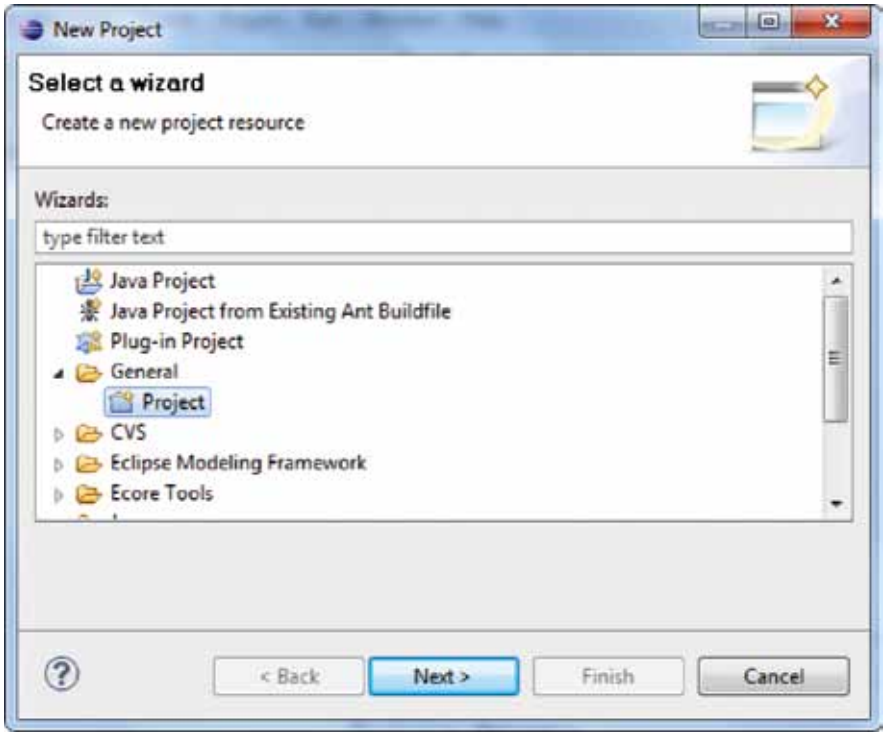


Abbildung 13: Anlegen eines neuen Projekts

Elemente bearbeitete werden (siehe Abbildung 16). Änderungen, die im Beispieleditor gespeichert sind, sind in der Datei „myLeague.bowling“ im XMI-Format serialisiert. Das Ergebnis kann eingesehen werden, wenn die Datei mit einem Text-Editor geöffnet wird.

Fazit

Dieser Teil der Reihe „Eclipse-Modeling-Framework“ beschreibt den direkten Weg vom ersten Modell zum generierten Code. Das Modell kann direkt getestet werden, indem man über den mitgenerierten Beispiel-Editor Instanzen des Modells anlegt. Auf diese Weise kann ein Modell iterativ aufgebaut werden, indem es nach jeder Erweiterung neu generiert wird. In der nächsten Ausgabe beschreiben wir den Aufbau der generierten Plug-ins sowie die Verwendung der API von Eclipse, beispielsweise um neue Instanzen des Modells programmatisch zu erzeugen. Anschließend beschreiben wir zusätzliche Technologien, die mit einem EMF-Modell verwendet werden können, beispielsweise eine Server-Lösung für EMF-Modell-Instanzen. Bei Fragen zum Artikel oder zu EMF im Allgemeinen können Sie gern die Autoren unter [5] kontaktieren.

Links

[1] http://www.eclipse.org/emf
[2] http://www.eclipse.org/downloads/
[3] http://eclipsesource.com/blogs/wp-content/uploads/2011/03/exampleSolution.zip
[4] http://eclipsesource.com/emftutorial
[5] jhelming@eclipsesource.com

Jonas Helmig
jhelming@eclipsesource.com

Maxumilian Kögel
mkoegel@ eclipsesource.com

Jonas Helming und Maximilian Kögel sind Eclipse-EMF/RCP-Trainer und Consultants sowie Geschäftsführer der EclipseSource München GmbH. Sie leiten die Eclipse-Projekte „EMFStore“ und „EMF Client Platform“.

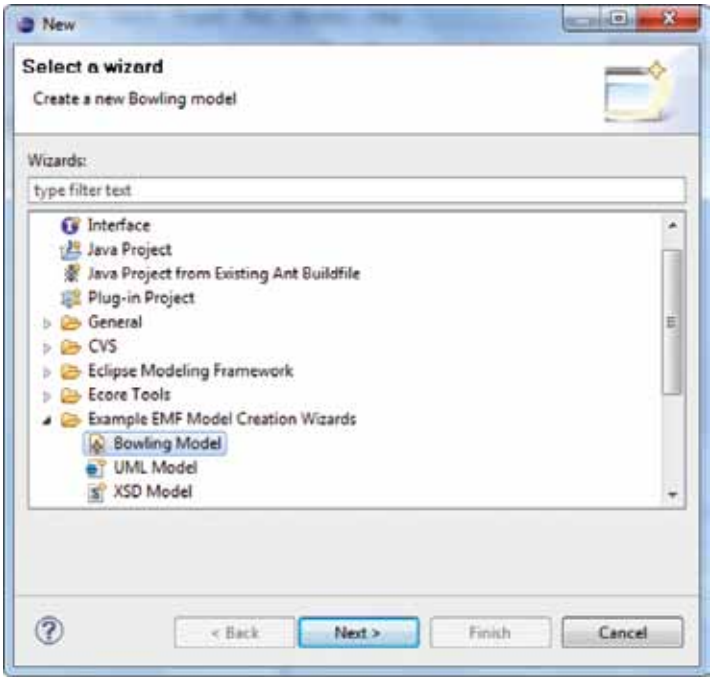


Abbildung 14: Erzeugen einer Modell-Instanz

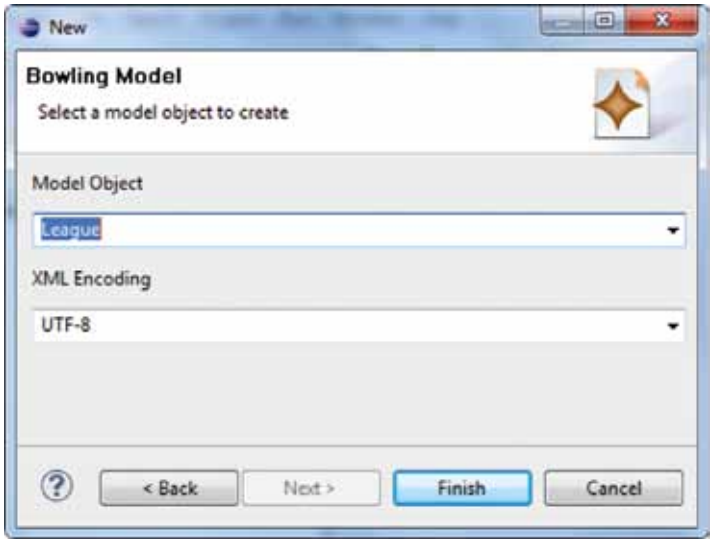


Abbildung 15: Auswahl des Root-Modell-Elements

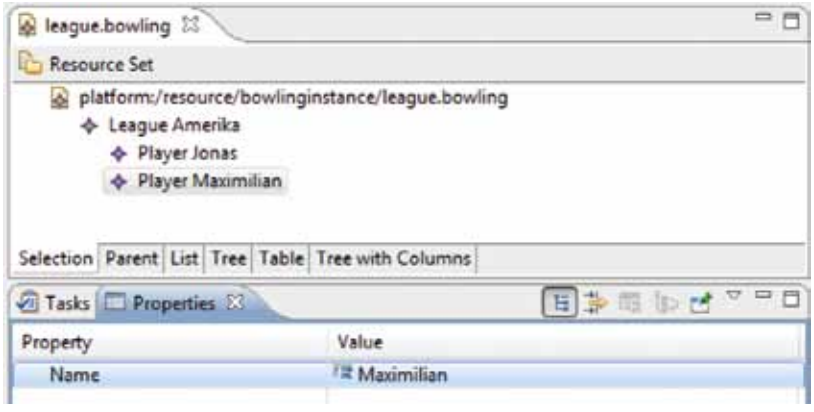


Abbildung 16: Der generierte Beispieleditor

Bereit für Wolke sieben – was die Oracle Public Cloud kann

Robert Szilinski und Michael Krebs, esentri consulting GmbH

Das Schlagwort „Cloud" mit seinen unterschiedlichsten Ausprägungen ist wahrlich kein neues Phänomen, doch es scheint so, als wäre das Thema „Cloud Computing" aktuell präsenter denn je. Sogar in der Fernsehwerbung findet man großangelegte Kampagnen zu Apples „iCloud", dazu gesellt sich im Vorabendprogramm die Telekom-Cloud und auch im B2B-Sektor hat es den Anschein, als wäre der „Krieg der Wolken" ausgebrochen. Dieser Artikel zeigt, was Oracle mit seiner „Public Cloud" zu bieten hat und wo die fundamentalen Unterschiede zu anderen Plattformen im Hinblick auf die technischen und funktionalen Anwendungsgebiete liegen.

Wir beginnen mit der Frage der grundsätzlichen Architektur und ihrer Auswirkungen auf die Sicherheit der Daten, denn wenn man über Cloud-Umgebungen diskutiert, steht die Datensicherheit meist als erster Punkt in der Liste der entscheidenden Faktoren bei der Evaluierung einer möglichen Cloud-Strategie. Im Gegensatz zur Datenhaltung im eigenen Rechenzentrum besteht noch immer eine große Unsicherheit und eine oft unbegründete Angst, dass die volle Kontrolle über die eigenen Daten mit der Verwendung von Cloud-Diensten verloren geht und man sich damit der Zuverlässigkeit eines Dienstleisters ausliefert, der weitab des eigenen Verfügungsbereichs agiert. Hierbei stehen sich in den letzten Wochen vor allem Salesforce und Oracle gegenüber, die mit unterschiedlichen Cloud-Konzepten aufwarten. Die technischen Möglichkeiten und Unterschiede verschwinden dabei meist jedoch in einer Flut von Marketing-Begriffen, die jede Menge Interpretations-Spielraum zulassen. In dieser Diskussion spielt vor allem auch der Begriff „Multi-Tenancy" eine entscheidende Rolle, der jedoch tatsächlich einen feinen, aber bedeutenden Unterschied im Architektur-Ansatz ausmacht (siehe Abbildung 1).

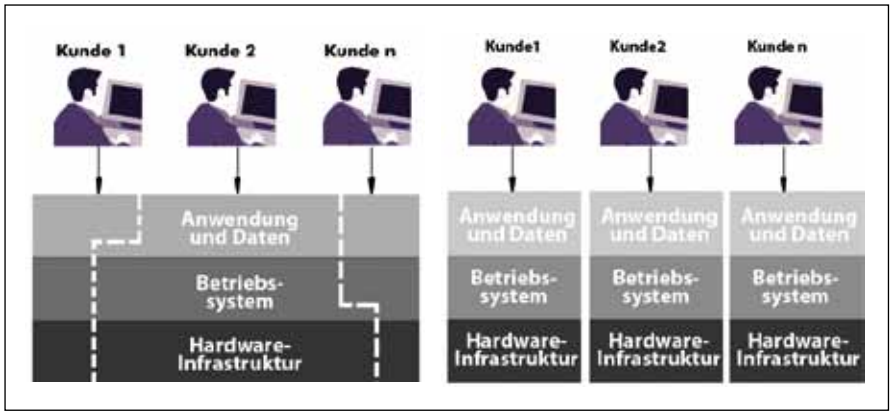


Abbildung 1: Multi-Tenancy vs. Single Tenancy

Ist „Multi-Tenancy" Voraussetzung für die „wahren" Clouds?

Bei einer Multi-Tenancy-Architektur wird nicht für jeden einzelnen Kunden eine dedizierte Infrastruktur bereitgestellt. Alle Kunden nutzen stattdessen eine zentrale Plattform, was häufig auch als ein Grundmerkmal für die Nutzung der Terminologie „Cloud-Computing" gesehen wird. Die angebotene Plattform und die darauf zur Verfügung gestellten Applikationen selbst sind dabei in virtuelle Partitionen unterteilt. Jedem Kunden steht eine virtuelle Anwendungs-Instanz zur Verfügung. Diese Multi-Tenancy-Architekturen sind demzufolge die Grundlage von „Software as a Service"-Diensten, denn nur hierdurch

kann die gesamte Kapazität der Plattform dynamisch für alle Mandanten bereitgestellt und genutzt werden.

Zusätzlich sind Updates oder Upgrades der einzelnen Komponenten problemlos möglich, da keine Einzelbetrachtung von kundenspezifischen Instanzen möglich ist. Es gilt der Grundsatz, dass die Plattform und die zur Verfügung gestellten Funktionalitäten für alle Kunden immer den gleichen Stand haben. Doch neben den Vorteilen liegt genau an dieser Stelle auch das größte Problem dieses Ansatzes: Die kundenspezifischen Anwendungs-Instanzen sind eben nur virtuell vorhanden, physisch befinden sich alle Anwendungen und die zugehörigen Daten in einem „großen Topf", was gerade bei sensiblen und unternehmenskritischen Anwendungen und Daten oft als eklatantes Sicherheitsrisiko betrachtet wird. Denn wer stellt bei all der Virtualisierung sicher, dass nicht doch ein anderer Benutzer oder der Anbieter selbst über den Tellerrand hinaus in den eigenen Datentopf spickt? Zumindest sind das die Bedenken.

„Single Tenancy" – sicher, abgeschlossen, aber noch wirkliches „Cloud-Computing"?

Als Alternative zu den Multi-Tenancy-Architekturen finden sich in der Cloud sogenannte „Single-Tenancy-Konzepte". Hierbei steht jedem Kunden seine dedizierte und als angeschlossenes Silo zu betrachtende Plattform inklusive reservierter Infrastruktur zur Verfügung. Die individuellen Sicherheits- und Konfigurationsmöglichkeiten sind dadurch mit einer lokalen Infrastruktur zu vergleichen. Doch gehen hierbei natürlich elementare Vorzüge der Cloud-Philosophie verloren, so sind zum Beispiel Updates immer auf jeder einzelnen Instanz durchzuführen, weshalb man bei Single-Tenancy-Angeboten eher von einem Outsourcing der Infrastruktur als von wirklichem Cloud-Computing sprechen kann.

Gerade bei modernen „Software-as-a-Service"-Angeboten scheint dieser Ansatz nicht dem wirklichen Kerngedanken des Cloud Computing zu entsprechen, bei dem Ressourcen und Kapazitäten gebündelt zur Verfügung stehen und den einzelnen virtuellen Instanzen dynamisch bereitgestellt werden. Dadurch kann jeder Teilnehmer ohne dediziert zu reservierende Leistung von der vollen Performance der Cloud-Plattform profitieren. Darüber hinaus können Applikationen schnell und unkompliziert einer breiten Masse an Teilnehmern zur Verfügung gestellt werden, da bei den Anwendungen selbst nur die Mandantenfähigkeit sichergestellt werden muss, die Daten und die Plattform aber zentral genutzt werden können.

Die Oracle Public Cloud macht durch Virtualisierung Single Tenancy „Cloud Ready"

Die neue Public Cloud von Oracle [1] ist als eine virtualisierte Single-Tenancy-Plattform gestaltet (siehe Abbildung 2). Hierdurch wird den Kunden eine private Infrastruktur und vor allem eine dedizierte Datenbank zur Verfügung gestellt, die für jeden Kunden einen isolierten und abgeschotteten Bereich bietet und somit in Sachen Sicherheit punkten kann. Salesforce.com setzt mit seiner Cloud-Infrastruktur dagegen mit der Bereitstellung seiner Applikationen kompromisslos auf „Multi-Tenancy"-Architektur.

Abbildung 2: Gesamt-Architektur der Oracle Cloud

Ein Kernaspekt, den Oracle in seiner neuen Cloud-Architektur in den Vordergrund stellt, sind die in der Oracle Public Cloud verwendeten Industriestandards. Natürlich ist dieser Begriff mit Vorsicht zu genießen, denn der erwähnte Standard wird zum Teil einzig und allein durch Oracle selbst definiert. Trotzdem steckt dahinter mehr als nur Marketing. Will man die Vorzüge anderer Cloud-Anbieter nutzen, die sowohl Infrastruktur als auch Plattform-Dienstleistungen anbieten, setzt dies meist einen radikalen Wandel der eigenen Infrastruktur und IT-Strategie voraus, da eine Kombination von bestehenden Systemen mit den eher visionär und in sich geschlossenen und proprietären Cloud-Plattformen nur bedingt möglich ist. JEE-Anwendungen für die Oracle Public Cloud können jedoch genauso lokal wie in der Cloud betrieben werden und es ist somit möglich, auch schrittweise in Richtung Cloud zu gehen.

In der Oracle Public Cloud ist die „Infrastructure as a Service" durch Oracles Exadata beziehungsweise Exalogic zur Verfügung gestellt. Dabei wird auch das gesamte Management der Infrastruktur, wie in Cloud-Plattformen üblich, durch Oracle übernommen. Neben der Oracle-Datenbank 11*g* gehört dazu natürlich auch der Oracle WebLogic Server 11*g*, der quasi auf Knopfdruck eine lauffähige Runtime-Umgebung für JEE-Applikationen zur Verfügung stellen kann, die alle gängigen Standards unterstützt und somit weit jenseits von proprietären Cloud-Plattformen anzusiedeln ist.

JEE in der Cloud

Eine zentrale Rolle bei Oracle spielt nicht erst seit der Übernahme von Sun die Java-Plattform, die heute mehr denn je ein entscheidender und vor allem unverrückbarer Baustein in der Oracle-Strategie geworden ist. So ist es kein Wunder, dass sämtliche neuen Fusion Applications ausschließlich auf dem JEE-Stack entwickelt werden. Dabei kommt dem Oracle Application Development Framework (ADF) [2] eine zentrale Rolle zu, was bisher außerhalb des Oracle-Ökosystems nicht wirklich der Fall war. Mit der Public Cloud könnte sich dies jedoch ändern, da durch die Verwendung von Oracles WebLogic Server 11*g* keine zusätzlichen Lizenzen für ADF anfallen und es somit vielleicht auch für Entwickler interessant ist, die bisher ADF zusätzlich hätten lizenzieren müssen. ADF selbst basiert auf den aktuellen JEE-Standards und so ist es nicht überraschend, dass die Oracle Public Cloud diese auch ohne ADF unterstützt. Im Wesentlichen sind dies:

- JEE 5 Support
- Servlet 2.5
- JSP 2.1
- JSF 2.0
- EJB 2.1 und 3.0 mit lokalen Interfaces

- JAX-WS 2.1 (aktuell noch beschränkt auf Inbound-Services)
- JAX-RS 1.1 für REST Services

Oracle-Datenbank 11g mit JPA-2.0-Unterstützung

Für die Anbindung der Oracle -Datenbank 11g steht JPA 2.0 mit Unterstützung von JDBC 4.0 zur Verfügung. Dabei können zunächst nur Datenbank-Verbindungen verwendet werden, die innerhalb der Cloud-Service-Gruppe liegen. Diese werden dann aber automatisch als JDBC Datasource hinzugefügt. Möglich ist auch die Verwendung von EclipseLink 2.1.1.

In Kombination von Oracles WebLogic Servern 11g und den Oracle-Datenbanken entstehen durch die Public Cloud auch dynamische und flexible Entwicklungsumgebungen. Hierfür sind aber keine neuen Entwicklungstools notwendig, sondern es werden zukünftig die beliebten IDEs wie JDeveloper, NetBeans oder Eclipse unterstützt. Gerade für letzteres wurde in den vergangenen Tagen das Oracle Enterprise Pack for Eclipse (OEPE) in Version 11.1.1.8 [3] veröffentlicht. Darin enthalten ist ein erster Vorgeschmack darauf, wie einfach die Oracle Public Cloud in die IDE integriert werden wird. So gibt es nun einen Assistenten, mit dem sich „Oracle Public Cloud Web Projects" erstellen lassen. Das Deployment in die Cloud wird ebenfalls von der IDE übernommen, sofern die Zugangsdaten richtig hinterlegt werden.

Angekündigt ist in diesem Zusammenhang auch ein Cloud SDK, mit dem die Dienste der Oracle Public Cloud angesprochen werden können. So wird durch den Whitelisting-Mechanismus mithilfe des Cloud SDK geprüft, ob nur APIs, die auch von der Cloud unterstützt werden, verwendet wurden.

Betrieb und Security

Auch der Betrieb der Cloud soll sich deutlich vereinfachen. Logfiles werden beispielsweise automatisch rotiert und das System kann sowohl über Oracles Enterprise Manager im Web überwacht werden als auch über Command-Line-Interfaces oder REST-Schnittstellen.

Im Bereich der Security von Anwendungen nutzt Oracle die bewährten Webservice-Security-Lösungen (WS-Security 1.0 und 1.1 kompatibel) und stellt für alle Anwendungen in der Service-Gruppe sein Single-Sign-on-Konzept zur Verfügung. Weiterhin werden alle Deployment-Archive automatisch auf Viren gescannt, um etwaige Sicherheitsrisiken zu vermeiden.

SaaS in der Praxis: Oracle Fusion Applications und Oracle Social Network

Applikationen, die als Software as a Service flexibel über Mietmodelle bereitgestellt werden, sind spätestens seit salesforce.com ein maßgeblicher Wachstumstreiber von Cloud-Anbietern. Auch Oracle setzt mit seiner neuen Fusion Application Suite auf ein „Software as a Service"-Modell und bietet Zug um Zug alle Applikationen der Fusion Applications auch als Mietmodell an.

Neben den bereits in der Cloud erhältlichen Modulen aus den Bereichen „CRM", „Human Resources" und „Business Intelligence" erhalten auch neue Applikationen wie das eigene Oracle Social Network Einzug in die Cloud. Mit diesem Werkzeug bietet Oracle eine vollintegrierte Kommunikationsplattform an, welche die Kollaboration im Sinne sozialer Netzwerke im Kern der Unternehmen verankern soll. Darüber hinaus sind auch alle Applikationen als On-Premise-Varianten erhältlich. Genau hier ist auch einer der entscheidenden Unterschiede in der Strategie von Oracle im Vergleich zu Salesforce zu finden, die dieses Modell nicht anbieten.

Fazit

Durch die Kombination von bewährten JEE-Standards mit dem „Single-Tenancy"-Gedanken in der Public Cloud bietet Oracle eine hervorragende Möglichkeit der Variabilisierung von IT-Infrastrukturen. Die Nachteile einer „Single-Tenancy"-Architektur werden durch die Virtualisierung in der Oracle Cloud obsolet; das ebnet den Weg in die Wolke auch für Unternehmen, die bisher aufgrund von Sicherheitsbedenken oder gesetzlichen Vorgaben eine Nutzung von Cloud-Angeboten konsequent ablehnten.

Oracles Public Cloud ist keine Einbahnstraße: JEE-Applikationen können sowohl von lokalen Systemen in die Cloud portiert werden als auch umgekehrt. Während man sich bei vielen anderen Cloud-Plattformen nach dem initialen Wechsel nur noch innerhalb dieser Angebote bewegen kann und auch alle Zusatz-Entwicklungen rund um die genutzten Applikationen in der vom Anbieter vorgegebenen und meist proprietären Sprache erstellen muss, hat man bei Oracle zu jedem Zeitpunkt die Wahl zwischen Cloud, lokalem Deployment oder auch einer Kombination aus beiden Varianten.

Die beiden Autoren freuen sich jedenfalls schon auf den ersten echten Testdrive mit der Oracle Public Cloud und sind gespannt, wie sich dieses Konzept gegen bereits etablierte Anbieter durchsetzen wird.

Weitere Informationen

[1] Oracle Public Cloud: http://cloud.oracle.com

[2] Oracle ADF: http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html

[3] Oracle Enterprise Pack for Eclipse: http://www.oracle.com/technetwork/developer-tools/eclipse/overview/index.html

Robert Szilinski
robert.szilinski@esentri.com
Michael Krebs
michael.krebs@esentri.com

Michael Krebs beschäftigt sich als Head of Business Development bei esentri mit den aktuellsten Entwicklungen im Enterprise-Social-Networking. Eines seiner Kernthemen ist der Vergleich unterschiedlicher Cloud-Strategien über Anbietergrenzen hinweg.

Robert Szilinski berät Unternehmen seit vielen Jahren im JEE- und Oracle-Umfeld. Gemeinsam mit Andreas Badelt leitet er bei der DOAG die Special Interest Group Java und hält regelmäßig selbst Vorträge zu den Oracle Middleware-Technologien.

JCR in der Praxis mit Apache Jackrabbit und Spring

Dominic Weiser, esentri consulting GmbH

Durch den Erfolg von Online-Diensten wie Dropbox & Co. werden die Methoden zur Speicherung von Dokumenten in der Cloud wieder kontrovers diskutiert. Jenen, die nur ein paar Dokumente einfach und schnell online verwalten möchten, mag die aktuelle Dropbox Java API ausreichen. All jenen, die aber weitere Anforderungen wie die Versionierung von Dokumenten oder das Abbilden von Berechtigungssystemen haben, steht mit dem Java Content Repository (JCR) Standard eine mächtige Alternative zur Verfügung, die inzwischen von nahezu allen großen Herstellern in ihren Produkten unterstützt wird.

Der Artikel zeigt, wie sich der JCR-Standard in der Praxis umsetzen lässt und wie man mit wenig Entwicklungsaufwand ein eigenes Java Content Repository aufsetzen und integrieren kann. Als Beispiel dienen Apache Jackrabbit [1] und das Spring Framework.

Ein Blick zurück

JCR ist das Ergebnis des 2005 fertiggestellten Java Standard Requests (JSR-170). Bereits 2009 wurde Version 2 im JSR-283 beschlossen. Im Grunde lässt sich JCR in zwei Funktionsbausteine aufteilen. Im ersten sind alle Mechanismen definiert, die den lesenden Zugriff auf ein Content Repository ermöglichen. Unter anderem zählen dazu:

- *Authentifizierung und Autorisierung*
  Dieser Mechanismus unterliegt vornehmlich dem Java Authentication and Authorisation Service (JAAS). Mithilfe des Workspace und der Benutzer-Credentials kann sich der Anwender an dem Content Repository anmelden und erzeugt so eine individuelle Session. Diese ist der Kern des Arbeitens mit einem Content Repository.
- *XPath-Unterstützung*
  Der Aufbau des Content Trees in JCR ist ähnlich dem Aufbau eines XML-Dokuments. Aus diesem Grund wurde der XPath-Standard für JCR adaptiert. Über die Session kann jede beliebige XPath Query auf das Content Repository angewendet werden. Seit Version 2 ist die XPath-Unterstützung „deprecated". Seither wird das Suchen besser und effektiver mittels Apache Lucene unterstützt.
- *Über Knoten und Eigenschaften iterieren*
  Dazu werden entweder der direkte Pfad zu einem Objekt angegeben oder Inhalte über XPath Querys ermittelt.
- *Lesender Zugriff auf Eigenschaften*
  JCR ist so aufgebaut, dass alle Inhalte in Eigenschaften abgespeichert werden. Dabei wird nicht zwischen Nutz- oder Metadaten unterschieden. So werden beispielsweise die Binärdaten eines Bildes genauso in Eigenschaften abgespeichert wie Informationen zum technischen Ersteller.

Im zweiten Baustein werden auch schreibende Zugriffsmethoden zur Verfügung gestellt:

- *Hinzufügen, Bearbeiten und Entfernen von Knoten und Eigenschaften*
  Über das Session-Objekt sind die nötigen Schnittstellen erreichbar, um Inhalte zu verändern, neue hinzuzufügen oder auch zu entfernen. Die Session ist deshalb zentral, weil darüber die nötigen Berechtigungen erzeugt und überprüft werden. Mittels JAAS können auch hierfür sehr feingranulare Berechtigungskonzepte definiert werden.
- *Zuweisung von Typen zu Knoten*
  Wie bereits oben erwähnt, gibt es in JCR keine Trennung zwischen Nutz- und Metadaten. Dies führt konzeptuell dazu, dass es nötig ist, Knoten voneinander zu trennen. Dazu verfügt jeder Knoten über einen Typ, der auch über Vererbungen zugewiesen werden kann. Er gibt zum Beispiel Auskunft über enthaltene Daten oder die Einteilung in Kategorien.

Weiterhin bietet der JCR-Standard eine weitreichende Unterstützung für folgende Operationen an:

- *Versionierung*
  Das Repository lässt sich so konfigurieren, dass Änderungen an Inhalten versioniert werden. Dadurch können ältere Versionen eines Inhalts wiederhergestellt werden. Das Versionierungs-System wurde gemäß dem Workspace Versioning and Configuration Management (WVCM) API [2] Standard entwickelt.
- *Locking*
  Besonders Cloud-Anwendungen sind darauf ausgelegt, zeitgleich von vielen

Anwendern benutzt zu werden. Gerade bei der Manipulation von Daten können sehr schnell Konflikte entstehen, die über den integrierten JCR-Locking-Mechanismus vermieden werden können.

- *Transaktionen*
  Das Transaktionsmanagement mittels Java Transaction API [3] erlaubt es dem Entwickler selbst zu bestimmen, wann Änderungen persistiert werden.

Apache Jackrabbit

Es stellt sich nun natürlich die Frage, wie man selbst schnell ein eigenes Repository aufsetzen und testen kann. Neben den etablierten Herstellern von Content-Management-Systemen gibt es hierfür auch freie Alternativen. Für diesen Artikel wurde die JCR-Referenz-Implementierung Apache Jackrabbit gewählt. Als Open-Source-Projekt wird Jackrabbit unter der Apache-Lizenz 2 zur Verfügung gestellt. Dabei unterstützt Jackrabbit auch die optionalen Mechanismen des JCR-Standards wie Locking, Versioning und Transaktionen. Zusätzlich verfügt die Apache-Implementierung über weitere sehr interessante Features wie eine Volltextsuche oder eine integrierte WebDAV-Schnittstelle.

Das Aufsetzen eines eigenen Repository an sich ist relativ schnell erledigt, sobald die Frage geklärt ist, wie das Deployment-Szenario aussehen soll. Die wohl einfachste Möglichkeit ist die direkte Integration in die Applikation, wie Abbildung 1 zeigt.

Deployment-Option 1: Direkte Integration

Dazu muss die Jackrabbit Library mit allen Dependencies in die Applikation integriert werden. Im einfachsten Fall verwendet man den von Jackrabbit direkt bereitgestellten Mechanismus und lässt sich ein Repository automatisch erzeugen (siehe Listing 1).

Beim ersten Initialisieren wird so ein lokales Repository erzeugt, das die Jackrabbit-Standardeinstellungen hat. Zu diesen zählt auch ein JAAS-Security-Provider, der drei Benutzergruppen von Haus aus unterscheidet:

1. Anonym (nur lesender Zugriff)
2. User
3. Admin

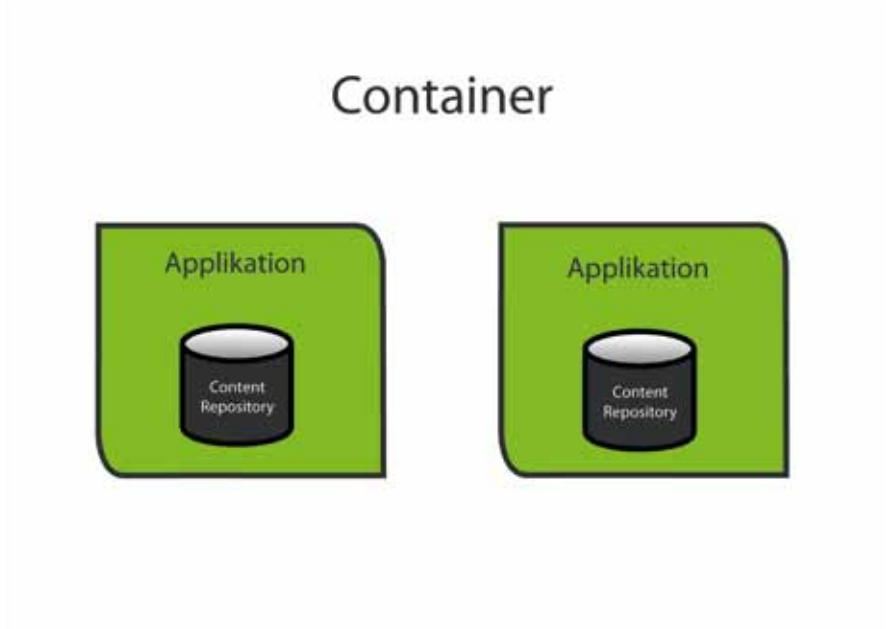


Abbildung 1: Content Repository in der Applikation

```
import javax.jcr.Repsitory;
import org.apache.jackrabbit.core.TransientRepository;
...
Repository repository = new TransientRepository();
```

Listing 1: Initialisierung eines Standard-Repository

```
<Repository>
  <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
    <param name="path" value="${rep.home}/repository"/>
  </FileSystem>
  <Security appName="Jackrabbit">
    <AccessManager class="org.apache.jackrabbit.core↵
    .security.SimpleAccessManager">
    </AccessManager>
    <LoginModule class="org.apache.jackrabbit.core.security.SimpleLoginModule">
      <param name="anonymousId" value="anonymous"/>
      <param name="adminId" value="admin"/>
    </LoginModule>
  </Security>
  <Workspaces rootPath="${rep.home}/workspaces" defaultWorkspace="default" />
  <Workspace name="${wsp.name}">
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
        <param name="path" value="${wsp.home}"/>
    </FileSystem>
    <PersistenceManager class="org.apache.jackrabbit.core.state↵
    .xml.XMLPersistenceManager" />
  </Workspace>
  <Versioning …/>
</Repository>
```

Listing 2: Jackrabbit-Konfigurations-XML

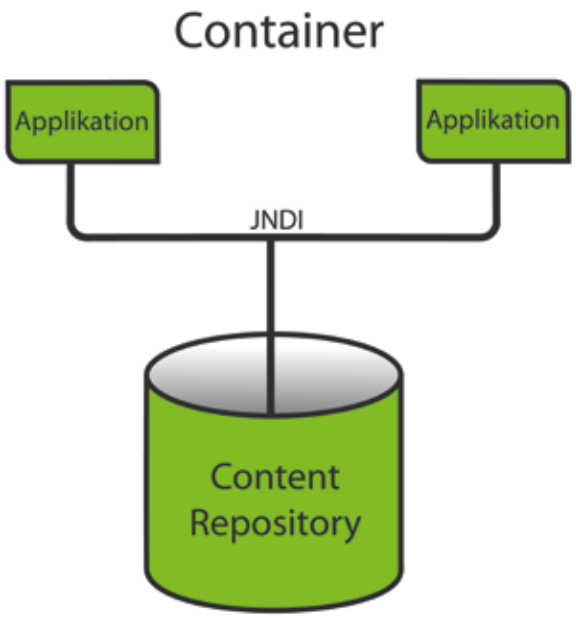


Abbildung 2: Ein Content Repository für mehrere Applikationen

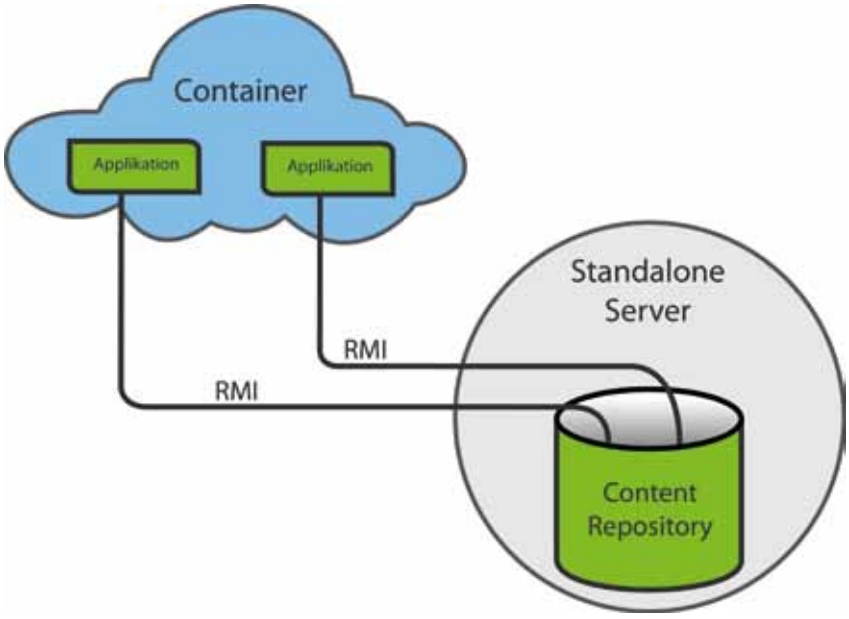


Abbildung 3: Content Repository als Standalone Server

```
<GlobalNamingResource>
  <Resource name="jcr/globalRepository" auth="Container"
    type="javax.jcr.Repository"
    factory="org.apache.jackrabbit.core.jndi.BindableRepositoryFactory"
    configFilePath="{pfadZumConfigFile}"
    repHomeDir="{repositoryHomeVerzeichnis}" />
</GlobalNamingResource>
```

Listing 3: JNDI Eintrag server.xml (Tomcat)

Feingranulare Berechtigungskonzepte lassen sich einfach in der zentralen Konfigurationsdatei definieren (siehe Listing 2). Neben dem Security-Modell sollten hier auch der Speichertort des Repository und der Suchindexe sowie die Konfigurationen der Versionierung eingestellt werden.

Das Beispiel in Listing 2 definiert ein File-basiertes Repository. Alternativ könnte man an dieser Stelle auch eine Datenbank für die Persistierung angeben. Dieses Repository wird über eine Beispiel-Implementierung des JAAS-Standards (Zeilen 5 bis 13) abgesichert. Wer stattdessen einen eigenen Security-, Access- oder Login-Manager benutzen möchte, muss diesen hier entsprechend angeben. Zusätzlich wird definiert, welcher Workspace (Zeile 15) verwendet wird und wo sich dieser befindet (Zeile 14). Dabei stellt das Beispiel für das bessere Verständnis nur einen minimalen Ausschnitt dar. Weitere Aspekte wie Clustering, DataStore und Suchindexe wurden hier nicht definiert, weil diese gut dokumentiert (siehe [1] unter Jackrabbit Configuration) sind.

Deployment-Option 2: Zentrales Repository mit JNDI-Zugriff

In der zweiten Deployment-Variante wird das Repository zentral installiert und kann via Java Naming and Directory Interface (JNDI) [4] angebunden werden. Im Gegensatz zum integrierten Repository können sich so mehrere Anwendungen ein Repository teilen (siehe Abbildung 2). Listing 3 zeigt, wie die JNDI-Definition für einen Tomcat 6 Server aussieht.

Die Pfadangaben zum configFilePath (Zeile 5) und repHomeDir (Zeile 6) variieren je nach Anwendung. Das Repository startet zusammen mit dem Server.

Um von der Applikation auf das Repository zugreifen zu können, muss die Referenz mit bekannten Mechanismen in der web.xml (der Applikation) als Ressource bekannt gemacht werden (siehe Listing 4). Sind diese Einstellungen korrekt definiert worden, kann das Repository aus der Applikation referenziert werden (siehe Listing 5).

Deployment-Option 3: Zentraler Zugriff mittels RMI

Remote Method Invocation (RMI) [5] ist die flexibelste Variante, da alle Deployment-

```
<resource-env-ref>
  <description>Content Repository</description>
  <resource-env-ref-name>jcr/globalRepository</resource-env-ref-name>
  <resource-env-ref-type>javax.jcr.Repository</resource-env-ref-type>
</resource-env-ref>
```

Listing 4: Referenz-Zeiger web.xml

```
import javax.naming.Context;
import javax.naming.InitialContext;
...
InitialContext context = new InitialContext();
Context localContext = (Context) context.lookup(„java:comp/env“);
Repository repository = localContext.lookup(„jcr/globalRepository“);
```

Listing 5: Initialisierung Content Repository

```
import javax.jcr.Repository;
import org.apache.jackrabbit.rmi.repository.URLRemoteRepository;
…
Repository repository =
    new URLRemoteReporsitory(„http://localhost:8080/*jackrabbit_folder*/rmi“);
```

Listing 6: Verbinden eines Remote Repository via RMI

```
import javax.jcr.Node;
import javax.jcr.Session;
import javax.jcr.SimpleCredential;
import javax.jcr.query.*;
...
Session session = reposiory.login(
                                        new SimpleCredential(„admin“, „admin“.
toCharArray());
session.getRootNode().addNode(„HelloWorld“);
session.getNode(„HelloWorld“).setProperty(„message“, „HelloWorld“);
session.save();

QueryManager manager = session.getWorkspace().getQueryManager();
Query query = manager.createQuery(„SELECT * FROM nt:base“,Query.SQL);
QueryResult result = query.execute();
NodeIterrator iterrator = result.getNodes();
...
Node helloWorld = session.getNode(„HelloWorld“);
System.out.println( helloWorld.getProperty(„message“).getString() );
helloWorld.remove();
session.save();
```

Listing 7: Erzeugung eines HelloWorld-Knoten in JCR

Szenarien auf die gleiche Art und Weise definiert werden können. Zum Austausch muss lediglich die Konfiguration des Repository beziehungsweise die URL entsprechend angepasst werden.

Für das RMI-Beispiel wird der Standalone Server von Jackrabbit verwendet. Dieser kann direkt über die Jackrabbit Homepage heruntergeladen, lokal gestartet oder als .war-File „deployt“ werden. Ist der Server gestartet, kann er über die vorgegebene RMI-Schnittstelle erreicht werden (siehe Listing 6).

Mit diesen drei verschiedenen Adressierungsarten kann das Content Repository auf unterschiedliche Arten eingebunden werden. Welche Variante gewählt wird, hängt von den jeweiligen Bedürfnissen ab. Wenn nur eine kleine Applikation mit eigenem Content Repository benötigt wird, eignet sich die lokale Variante 1 am besten. Will man hingegen verschiedene Applikationen, die alle mit Zugriff auf ein Content Repository ausgestattet sind, auf dem gleichen Server laufen lassen, wählt man am besten die Anbindung via JNDI und „deployt“ das Repository direkt auf dem Server. Geht man davon aus, dass verschiedene Applikationen auf unterschiedlichen Servern auf das gleiche Repository zugreifen, sollte das dritte Deployment-Modell über RMI gewählt werden.

JCR in der Praxis

Ist die Verbindung zum Repository aufgebaut, geht es durchgängig und leicht verständlich weiter. Mit dem vorhandenen Repository kann sich der User anmelden und eine Session erzeugen. Innerhalb der Session können neue Inhalte erstellt, über bestehende Inhalte iteriert oder diese verändert werden.

Listing 7 zeigt, wie ein HelloWorld-Knoten dem bestehenden Content Repository hinzugefügt wird. Mit der Session wird der Root-Knoten bestimmt und zu diesem der neue Order (HelloWorld) hinzugefügt. Diesem Ordner wird die Nachricht „HelloWorld“ hinzugefügt. Die Änderungen werden erst durch den save()-Aufruf persistiert. Durch dieses JTA-basierte Transaktionsmodell ist es auch möglich, alle Änderungen bis zur Persistierung zu verwerfen.

Im zweiten Abschnitt wird beispielsweise eine Suche mittels JCR-SQL2 durch-

```
<beans ...>
<bean id="demoRepository"
        class="org.springframework.jndi.JndiObjectFactoryBean">
  <property name="jndiName">
    <value>java:comp/env/jcr/repository</value>
  </property>
</bean>
<bean id="springDemoController"
        class="com.companie.app.SpringDemoController">
  <property name="repository">
    <ref bean="demoRepository"/>
  </property>
</bean>
<bean id="urlMapping"
        class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="urlMap">
    <map>
      <entry key="/helloJcrSpring.htm">
        <ref bean="springDemoController"/>
      </entry>
    </map>
  </property>
</bean>
</beans>
```

Listing 8: Interaktion Spring und JCR

```
...
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class SpringDemoController implements Controller {
  private Repository repository;
  @Secured("DEMO_USER")
  public ModelAndView handleRequest(HttpServletRequest req,
        HttpServletResponse res) throws Exception {
  Session session = repository.login( ↵
    new SimpleCredentials(„admin", „admin".toCharArray()));
    ...
  }
  public void setRepository(Repository repository){
    this.repository = repository;
  }
}
```

Listing 9: SpringDemoController

geführt. Dabei unterscheidet sich die JCR-SQL2-Syntax von der allgemeinen SQL-Syntax. Seitdem XPath „deprecated" ist, werden alle Queries intern nach JCR-SQL2 übersetzt. Im Beispiel von Listing 7 wird ein Iterator erzeugt, der alle Knoten des Repository enthält.

Im dritten Abschnitt ist zu sehen, wie leicht man einen Knoten über den Namen referenzieren und in der Folge verändern kann. Danach lässt man die im ersten Abschnitt gespeicherte Nachricht ausgeben und löscht den Knoten anschließend mit all seinen Kindern. Auch hier muss das Löschen mittels „save()" persistiert werden.

Spring und JCR

Es bleibt nun noch zu klären, wie sich das beliebte Spring Framework für die Anbindung eines JCR Repository konfigurieren lässt und wie viel Entwicklungsaufwand dafür noch benötigt wird.

Entsprechend der Erwartungshaltung ist dies recht einfach möglich und geht leicht von der Hand. Glücklicherweise gibt es ein eigenes Spring-Modul, das die Interaktion mit einem Content Repository via JCR unterstützt und dem Entwickler bei der Erzeugung von Applikationen mit JCR-Anbindung zur Seite steht.

In den folgenden Listings soll kurz dargestellt werden, wie einfach die Repository-Konfiguration mittels Dependency Injection übergeben werden kann. Dadurch lassen sich, frei nach der Spring-Mentalität, Konfiguration und Logik sauber voneinander trennen.

Dies ermöglicht eine einfache und schnelle Austauschbarkeit einzelner Komponenten. Darüber hinaus wird gezeigt, wie die Anwendung zusätzlich mittels Spring Security abgesichert werden kann (siehe Listing 8).

Mit dem Öffnen der Seite /helloJcrSpring.htm wird der Aufruf via Spring-Beans an den springDemoController übermittelt. Diesem Controller wird das Repository per Dependency Injection übergeben (siehe Listing 9).

Dieser Controller wird durch den weitergeleiteten Aufruf geöffnet. Über die Methode setRepository wurde der Klasse via Dependency Injection das Repository übertragen. Wie man im Listing weiterhin sieht, ist auch die Verwendung eines existierenden Spring-Security-Modells möglich, und in unserem Fall können nur Benutzer der „DEMO_USER"-Rolle mit dem Repository interagieren.

Zwischenzeitlich wurde das Spring-JCR-Modul ausgelagert und wird als Spring Extension (SE-JCR) [6] weiterentwickelt. Leider stagniert diese Entwicklung seit Mai 2010 etwas. Es werden zwar weiter grundlegende Funktionalitäten zur Verfügung gestellt, jedoch fehlen die Neuerungen von JSR-283 alias JCR Version 2.

Es bleibt abzuwarten, wie SE-JCR in Zukunft weiterentwickelt wird und wann die fehlenden Funktionen nachgezogen werden.

Nichtsdestotrotz lässt sich die Spring Extension schon jetzt sehr gut einsetzen, auch wenn das Major Release 1.0 noch auf sich warten lässt.

Fazit

Abseits der Spring-Community hat sich JCR zu einem weitverbreiteten Standard gemausert. Man merkt sehr deutlich, dass ein großes Interesse der Community vorhanden ist. Ein Indikator dafür ist die massive Anzahl von Tutorials, Mailing-Listen, Foren-Einträgen und Artikeln rund um das Thema JCR. Gerade diese umfangreiche Dokumentation erleichtert den Einstieg enorm.

Weitere JCR-Implementierungen sind unter [7] zu finden. Wer sich über den JCR-Standard hinaus noch mehr Flexibilität wünscht, besonders in Richtung SOAP und REST-Schnittstellen, sei auf „Content Management Interoperability Service" (CMIS) verwiesen. Dieser erweitert unter anderem auch den JCR-Standard und wird bereits von vielen namhaften ECM-Herstellern wie IBM, Microsoft, SAP oder Oracle unterstützt.

Die große Nachfrage spiegelt sich auch darin wider, dass im Moment die JCR-Version 2.1 in der Entwicklung ist (JSR-333). Hier wurde bereits der Early Draft veröffentlicht und daraus geht hervor, dass vor allem einige Änderungen und Vereinfachungen an der API vorgenommen werden, um die ohnehin nicht sehr hohen Einstiegshürden nochmals zu senken.

Literatur

[1] http://jackrabbit.apache.org

[2] http://www.jcp.org/en/jsr/detail?id=147

[3] http://www.oracle.com/technetwork/java/javaee/jta/index.html

[4] http://www.oracle.com/technetwork/java/jndi/index.html

[5] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[6] https://jira.springsource.org/browse/SEJCR

[7] http://wiki.apache.org/jackrabbit/JcrLinks

Dominic Weiser

dominic.weiser@esentri.com

Dominic Weiser beschäftigt sich bei esentri mit der Entwicklung von Enterprise-Social-Networking-Lösungen. Sein Spezialgebiet sind SaaS-Anwendungen auf Basis von JEE und Spring.

Unbekannte Kostbarkeiten des SDK: Dynamic Proxy

Bernd Müller, Ostfalia – Hochschule für angewandte Wissenschaften

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Damit könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des SDK vorstellen: die unbekannten Kostbarkeiten.

Das Proxy-Pattern wird im Buch der „Gang of Four" wie folgt definiert: „Provide a surrogate or placeholder for another object to control access to it". Neben der reinen Stellvertreter-Funktionalität kann auch der Zugriff auf das Proxy, also Methodenaufrufe, kontrolliert werden. Wir demonstrieren in einem kleinen Beispiel sowohl die Stellvertreter-Funktionalität als auch die Zugriffskontrolle. Dazu bauen wir einen Entity-Manager nach JPA-Vorbild, jedoch ohne wirkliche Funktionalität nach und kontrollieren die CRUD-Methodenaufrufe für das Transaktionsmanagement.

Proxy und InvocationHandler

Die Basis für dynamische Proxies bilden die Klasse „Proxy" und das Interface „InvocationHandler" im Package „java.lang.reflect". Beide sind seit Version 1.3 im Java SDK enthalten und gehören zu den unbekannten Kostbarkeiten. Die Klasse „Proxy" besitzt die beiden statischen Fabrikmethoden „getProxyClass()" und „newProxyInstance()". Die erste Methode liefert eine dynamisch erzeugte Klasse, die eine Menge von Interfaces implementiert. Die zweite Methode gibt eine Instanz einer solchen Klasse zurück.

Nachdem nun die Implementierung des Proxies möglich ist, folgt der zweite Teil des Proxy-Patterns, die Delegation des Methodenaufrufs. Hierzu wird das Interface „InvocationHandler" verwendet. Beim Einsatz von „getProxyClass()" erfolgt dies explizit, bei der Verwendung von „newProxyInstance()" implizit als Parameter der Methode. Wir werden für ein Beispiel die zweite Alternative benutzen, da der zu erstellende Code kompakter ist.

Das Beispiel

Als Beispiel bauen wir JPAs „EntityManager" nach, beschränken uns aber auf die „persist()"-Methode und bleiben die tatsächliche Realisierung schuldig. Das Interface „EntityManager" ist also recht einfach (siehe Listing 1).

Eine Implementierung ist der „SimpleEntityManager" (siehe Listing 2).

```
public interface EntityManager {
  void persist(Object entity);
}
```

Listing 1

```
public class SimpleEntityManager implements
EntityManager {
   @Override
   public void persist(Object entity) {
        System.out.println(„Persisting „ +
        entity);
  }
}
```

Listing 2

Ziel ist nun, ein Proxy zu erzeugen, das dasselbe Interface implementiert und die Kontrolle des Methodenaufrufs erlaubt. Im Beispiel soll ein Transaktionsmanagement

eingebaut werden. Es sind aber auch andere Möglichkeiten, etwa das Ändern der Methodenparameter oder der Abbruch des Methodenaufrufs denkbar. Die Klasse „EMProxy“ implementiert das Interface „InvocationHandler“ und hat das zu vertretende Objekt als Instanzvariable (siehe Listing 3).

```
public class EMProxy implements Invoca-
tionHandler {
  private EntityManager em;
  public EntityManagerInvocationHandler
  (EntityManager em) {
      this.em = em;
  }
  ...
}
```

Listing 3

Das Interface besitzt „invoke()“ als einzige Methode (siehe Listing 4). Hier ist „proxy“ die Proxy-Instanz, für die die Methode aufgerufen wurde, „method“ die aufzurufende Methode und „args“ deren Parameter. Wir implementieren „invoke()“ wie in Listing 5 gezeigt.

```
Object invoke(Object proxy,
          Method method,
          Object[ ] args) throws Throwable
```

Listing 4

```
@Override
public Object invoke(Object proxy, Method method,
   Object[ ] args)
 throws Throwable {
   Object result;
   try {
     System.out.println(„hier Transaktion starten“);
     result = method.invoke(em, args);
   } catch (InvocationTargetException e) {
     System.out.println(„hier Transaktion zurückrollen“);
     throw e.getCause();
   }
   System.out.println(„hier Transaktion committen“);
   return result;
}
```

Listing 5

Auch hier haben wir analog zur Methode „persist()“ mit „println()“ angedeutet, wo die eigentliche Implementierung zu realisieren ist. Hier wird mit der Methode „Method.invoke()“ die konkrete Methode über Reflection aufgerufen und an den entsprechenden Stellen das Transaktionsmanagement implementiert. Die Verwendung des Proxies erfolgt nun über die Methode „EMProxy.createProxy()“ (siehe Listing 6).

```
EntityManager em = EMProxy.createProxy(
                new SimpleEntityManager());
   em.persist(...);
```

Listing 6

In diesem Beispiel erfolgt die explizite Angabe der realen Klasse. Man kann sich aber leicht Alternativen mit einer Fabrikmethode, die über System-Property gesteuert wird, oder – moderner – die Verwendung einer Annotation vorstellen (siehe Listing 7).

```
EntityManager em = EMProxy.newInstance();
// oder
@Inject
EntityManager em;
```

Listing 7

Fazit

Die Möglichkeiten zur Realisierung eines dynamischen Proxies sind bereits seit Version 1.3 fester Bestandteil des SDK, jedoch leider wenig bekannt. Da die Entscheidung, auf welche konkrete Implementierung gemappt wird, erst zur Laufzeit vorgenommen wird, sind sehr dynamische und flexible Systeme realisierbar.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

Anspruchsvolle Applikationen mit Adobe AIR realisieren

Ueli Kistler, Trivadis AG

Mit der Kombination von Web- und Desktop-Technologien ermöglicht Adobe AIR die Realisierung anspruchsvoller Desktop-Anwendungen. Der Artikel zeigt den Mehrwert von AIR bei der Anwendungsentwicklung.

Reine Web-Applikationen bieten oft nur eine unzureichende Funktionalität für den Endanwender. Will zum Beispiel der Finanzberater seine neueste Markt-Analyse veröffentlichen, kann er nicht einfach per „Drag & Drop" die Datei in den Browser ziehen. Er muss umständlich die Datei im Browserfenster suchen. Ist es dann soweit, kann er warten, bis die Datei hochgeladen wurde. Oder die Marketingfachfrau, die sich mit einer Desktop-Applikation beschäftigt, die leider immer noch nicht mit einem aktuellen Kundenstamm verknüpft ist, weil der Web-Service nicht ohne Weiteres in die Anwendung integriert werden kann. Solche und andere typische Anforderungen von Desktop-Anwendungen, die Interaktionen mit lokalen Anwendungen oder der Zwischenablage, „Drag & Drop", aber auch Remote-Kommunikation zum Beispiel mit Web-Services erfordern, können mit AIR effizient umgesetzt werden. Doch kann AIR die Lücke zwischen reiner Desktop- und Web-Anwendung schließen?

Adobe AIR

Adobe Integrated Runtime (AIR) ist eine Laufzeitumgebung für Rich Clients. Diese läuft eigenständig und außerhalb des Browsers. AIR ist Teil der Adobe-Flash-Plattform und basiert auf Technologien von Adobe Flash Player sowie der WebKit HTML Engine. Mit WebKit steht seit AIR 2.0 sogar eine HTML5-kompatible Webbrowser-Komponente zur Verfügung.

Anwendungen werden mit dem Adobe Flex SDK (Open Source) entwickelt, das sehr populär in der Rich-Internet-Application-Entwicklung ist, stetig weiterentwickelt wird und eine große Entwickler-Community besitzt. Eine Flex-Web-Anwendung für den Browser kann auch ohne großen Aufwand als eigenständige Desktop-Applikation eingesetzt werden.

Unterstützt werden Microsoft Windows, Mac OS X und Linux. Für mobile Plattformen wie Apple iOS oder Google Android steht mit Flex Mobile ein maßgeschneidertes Framework für Smartphones und Tablets zur Verfügung, das jedoch nicht direkt mit AIR vergleichbar ist und um Mobile-APIs für Kamera-Zugriff erweitert ist.

Flex für den Desktop

AIR basiert auf dem Adobe Flex SDK, das für die Entwicklung von RIA-Web-Anwendungen bereits eine Vielzahl von Komponenten bietet, und erweitert es um neue Elemente sowie APIs für den Desktop. Somit bringt es auch das nötige Rüstzeug, um auch komplexere Benutzeroberflächen effizient zu entwickeln. Dazu gehört auch eine native „Drag & Drop"-API.

Im Gegensatz zu AIR bietet eine Flex-Applikation nur „Drag & Drop" innerhalb der Anwendung, da der Browser keinen Zugriff auf den Desktop ermöglicht.

Mit nativem „Drag & Drop" auf allen unterstützten Plattformen können Daten in die Anwendungen importiert oder auch exportiert werden. Analog zur DragManager-Flex-API, welche innerhalb der Flex-Applikation verwendet wird, kann mit dem NativeDragManager auf „Drag & Drop"-Events des Systems reagiert werden.

Eine typische Anforderung von Desktop-Anwendungen ist das Ausführen einer externen Applikation. Häufig möchte auch einfach die zum Dateityp passende Applikation geöffnet werden. Beides wird seit

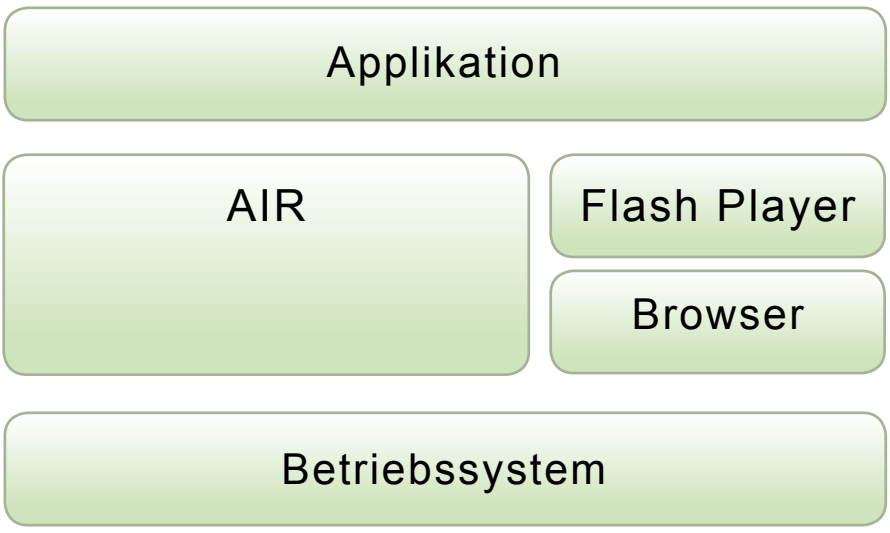


Abbildung 1: Laufzeitumgebung einer AIR-Applikation

Adobe AIR 2.0 mit der NativeProcess-API möglich.

Lese- und Schreibzugriffe auf das Dateisystem sind essentiell für Desktop-Applikationen. Bei einer Flex-Web-Anwendung wird vom Browser aus Sicherheitsgründen der Dateizugriff eingeschränkt oder verhindert. Deshalb unterscheidet sich das Verhalten der File-API mit AIR teilweise. Für einen Datei-Upload kann in der Web-Anwendung erst der Inhalt der Datei geladen werden, wenn vorher eine Benutzerinteraktion stattgefunden und der Benutzer eine bestimmte Datei ausgewählt hat. Da AIR wie jede andere Desktop-Anwendung Zugriff auf das Dateisystem hat, fallen diese Einschränkungen weg und Dateien können, wie sonst bei Desktop-Applikationen üblich, direkt bearbeitet werden.

AIR-Anwendungen werden deshalb in der Regel digital signiert, sodass der Benutzer vor der Installation der Anwendung entscheiden kann, ob er dieser Software vertraut. Für die Benutzeroberfläche bieten die FileSystem-Komponenten diverse Möglichkeiten, das Dateisystem ähnlich wie im Datei-Explorer zu visualisieren.

Seit AIR 2.0 können mit „File Promises" Dateien von einem Webserver sogar per „Drag & Drop" auf den Desktop verschoben werden. Nachdem die Dateien heruntergeladen wurden, sind diese auch entsprechend für das Betriebssystem so markiert, dass beim Öffnen eine Benutzerbestätigung erfolgen muss.

Integration mit Enterprise Services

Mit SQLite bietet Adobe AIR eine einfache relationale Datenbank, um Daten strukturiert abzuspeichern und Abfragen auszuführen. Das kann sich bei einer reinen Desktop-Applikation ohne Integration von Enterprise Services hilfreich sein, ist jedoch nur beschränkt geeignet für den Einsatz im Unternehmen. Durch die solide Basis, die das Flex SDK für die Applikationsentwicklung bietet, sind auch Integrationen bestehender REST- oder SOAP-Webservices problemlos möglich.

Mit Adobe BlazeDS können existierende Java-Services nahtlos mit der AIR-Applikation integriert werden. Der Aufruf eines Service, der bereits mit Java entwickelt wurde, ist so für den Applikationsentwickler völlig transparent, ähnlich wie bei RMI mit Java. BlazeDS ist Open Source und implementiert das binäre Action Message Format (AMF), das eine hohe Performance bei der Datenübertragung bietet.

Werden Web-Applikationen mit Spring entwickelt, können Spring Beans dank des Spring-Flex-Integration-Projekts sehr effizient als BlazeDS-Services exponiert werden. Zudem wird die Konfiguration von BlazeDS über die Spring-Konfiguration und die Integration mit Spring Security ermöglicht. AIR bietet auch eine umfassende Netzwerk-API, die TCP- und UDP-Sockets anbietet, um so auch auf Netzwerkdienste wie FTP zuzugreifen. Verschlüsselte Kommunikation ist dank TLS/SSL-Support möglich.

Dokumente drucken mit AIR

Eine Desktop-Applikation mit AIR kann sowohl PDFs erzeugen als auch drucken. PDFs lassen sich mit der AlivePDF-Open-Source-Bibliothek erstellen und dank einer integrierten WebKit-Komponente sogar direkt in der Applikation anzeigen. Das Drucken aus der Applikation ist mit Bitmap- und Vektordruck möglich. Im Vektormodus wird eine hohe Druckqualität insbesondere für Text erreicht, jedoch werden keine transparenten Grafiken unterstützt. Alternativ kann auch im Bitmap-Modus gedruckt werden, wodurch auch Transparenz-Effekte möglich werden. Die Druckqualität für Text sinkt jedoch im Vergleich zum Vektor-Modus ein wenig. Mit der FlexPrintJob-Klasse wird ein Job erstellt, der durch das Setzen des printAsBitmap-Flags in den Bitmap-Modus wechselt.

Einem Druckjob können beliebige Flex-Komponenten hinzugefügt werden. Ob die in der Benutzeroberfläche sichtbare Komponente gedruckt werden soll oder eine druckspezifische Ansicht besser geeignet ist, spielt für die Verwendung im Druckjob keine Rolle. In beiden Fällen wird eine Flex-Komponente gerendert und falls gewünscht auf die zu druckende Seite skaliert.

Eine druckspezifische Ansicht der Daten kann als gewöhnliche Flex-Komponente designt werden, mit allen Möglichkeiten, die Komponenten bieten. Das Styling kann mit CSS oder – seit Flex 4 – mit Skin-Klassen individuell angepasst werden. Falls eine andere Schriftart oder eine andere Positionierung der Elemente gewünscht ist, können sehr flexibel druckspezifische Darstellungen entwickelt werden (siehe Listing 1).

Ein paar Stolpersteine gibt es jedoch. In der Regel arbeiten viele Komponenten in Flex asynchron, so auch „Item Renderer", die verwendet werden, um Listeneinträge spezifisch darzustellen. Wenn beim Drucken eine Komponente gerendert werden muss, die beispielsweise externe Ressourcen wie Bilder asynchron lädt, kann es sein, dass beim Senden des Druckauftrags die Komponente noch nicht vollständig gerendert wurde und kein Bild erscheint. Um dieses Problem zu lösen, muss entweder mit „setTimeout" gearbeitet werden, um sicherzugehen dass die Komponenten vollständig gerendert wurden, oder die asynchron geladenen Ressourcen müssen synchron geladen werden.

Für mehrseitiges Drucken stehen im Moment vier Komponenten mit Paging-Funktionalitäten zur Verfügung wie die PrintAdvancedDataGrid-Komponente. Da Flex nur den sichtbaren Bereich rendert und

```
if (printJob.start() != true) {
 printJob.printAsBitmap = true;
 // Rendern
 FlexGlobals.topLevelApplication.addElement(myView);
 myView.visible = false;
 printJob.addObject(myview,
  FlexPrintJobScaleType.MATCH_WIDTH);
 // Gerenderte Komponente entfernen
 FlexGlobals.topLevelApplication.
  removeElement(myView);
 printJob.send();
}
```

Listing 1: Drucken mit Adobe AIR

auch druckt, würden bei der Tabellenkomponente „AdvancedDataGrid“ nur die ersten paar Zeilen auf dem Papier erscheinen Die PrintAdvancedDataGrid-Komponente bietet mit der Methode „nextPage()“ eine Möglichkeit, automatisch die nächsten Einträge der Tabelle zu laden, die auf der nachfolgenden Druckseite erscheinen sollen. Dabei wird die Dimension der Komponente automatisch berücksichtigt und die nächste Seite für den Druck gerendert.

Als Datenquelle kann eine bereits in der Benutzeroberfläche angezeigte AdvancedDataGrid-Komponente referenziert oder eine Collection per Data Binding verwendet werden. Für andere Elemente wie Listen existieren noch keine Komponenten, die explizite Paging-Funktionalitäten bieten würden.

Kompatibilität

AIR-Applikationen sind rückwärtskompatibel, das heißt es können für AIR 1.0 kompilierte Applikationen nach wie vor mit der aktuellen AIR-Laufzeitumgebung ausgeführt werden. Bei neuen AIR-Versionen können neue Funktionen dazukommen, die erst mit einer aktuellen Laufzeitumgebung unterstützt werden. AIR bietet deshalb die Möglichkeit, automatisch den Benutzer darauf hinzuweisen, dass die Laufzeitumgebung aktualisiert werden muss, wenn keine kompatible Version von AIR vorliegt.

Fazit

Mit AIR bietet Adobe eine umfangreiche Plattform zur Entwicklung von Desktop-Applikationen, die in Puncto Integration mit Enterprise-Services oder hinsichtlich Benutzerfreundlichkeit einer RIA-Web-Applikation in nichts nachstehen und die mit dem Desktop interagieren können. Die Lücke zwischen Web-und Desktop-Applikation wird jedoch nicht komplett geschlossen.

Ueli Kistler
ueli.kistler@trivadis.com

Ueli Kistler beschäftigt sich mit Java-Entwicklung und RIA-Themen. In seiner beruflichen Tätigkeit als Java EE Consultant bei Trivadis betreut er Kundenprojekte im Bankenbereich und vermittelt als Coach regelmäßigExpertenwissen.

„Java ist eine herausragende Technologie …“

Usergroups bieten vielfältige Möglichkeiten zum Erfahrungsaustausch und zur Wissensvermittlung unter den Java-Entwicklern. Sie sind aber auch ein wichtiges Sprachrohr in der Community und gegenüber Oracle. Wolfgang Taschner, Chefredakteur von Java aktuell, sprach darüber mit Andreas Haug, dem Vorsitzenden der Java User Group München.

Wie bist du zur Java User Group München gekommen?

Haug: Ich bin seit Langem Mitglied in der Gesellschaft für Informatik (GI) und als dann in der Regionalgruppe München der GI Überlegungen angestellt wurden, einen Arbeitskreis „Java“ zu gründen, habe ich mich dazu bereit erklärt ihn zu organisieren. Das war im Frühsommer 2000 und das erste Treffen der Java User Group München (JUGM) fand im Oktober 2000 statt.

Wie ist die Java User Group München organisiert?

Haug: Organisatorisch steht die JUGM auf ganz pragmatischen Füßen. Ich übernehme die Organisation der Meetings und wenn jemand Handlungsbedarf sieht und Lust hat sich einzubringen, erfolgen weitere Aktivitäten. So hat zum Beispiel Uli Bode die Herausgabe eines Java-Buches organisiert und Jürgen Thierack erledigt die Zusammenarbeit mit dem iJUG. Funktionieren kann das Ganze allerdings nur durch die Bereitschaft vieler Teilnehmer, selbst einmal einen Vortrag bei der JUGM zu übernehmen. Nur dadurch war es möglich, in den letzten elf Jahren mindestens elf Meetings pro Jahr abhalten zu können.

Was zeichnet die Java User Group München aus?

Haug: Unser Standortvorteil ermöglicht es uns, auf eine breite Basis an Java-Interessierten bauen zu können. Die einfache Struktur der JUGM unterstützt zusätzlich ein „Alles kann – Nichts muss“ – ich denke, das schätzen die Teilnehmer sehr.

Wie viele Abendveranstaltungen gibt es pro Jahr?

Haug: Wir planen mit einer Veranstaltung monatlich, wobei sich jedoch fast in jedem Jahr aus aktuellem Anlass Zusatztermine ergeben. In diesem Jahr werden es vermutlich fünfzehn Abendveranstaltungen werden.

Was motiviert dich denn besonders, als Vorstand die Java User Group München zu führen?

Haug: Ich denke, der Begriff „Vorstand“ ist etwas hoch gegriffen, Organisator trifft es eher. Es macht mir Spaß zu organisieren, wenn das entsprechende Interesse und die Bereitschaft der Mitglieder mitzuwirken vorhanden ist. Die Mitglieder entscheiden letztlich, wohin die Reise geht und wie sie mit Inhalten ausgefüllt wird. Die vielen abwechslungsreichen Vorträge aus verschiedenen Bereichen der Java-Technologie sind für mich sehr interessant und ich freue mich, dadurch einen sehr umfassenden Einblick in diese Technologien zu bekommen.

Andreas Haug, Vorsitzenden der Java User Group München

Was ist genau das Besondere an Java für dich?

Haug: Java ist die Programmiersprache, mit der ich in den letzten vierzehn Jahren überwiegend gearbeitet habe. Davor habe ich mich viel mit COBOL und C/C++ beschäftigt. Wenn ich darüber nachdenke, wie viele Diskussionen/Herausforderungen es damals gab, um beispielsweise einen plattformübergreifenden Build aufzubauen, oder wie viele Bibliotheken evaluiert/gekauft werden mussten, um Dinge zu tun, die die Programmiersprache Java schon enthält. Java war damals ein großer Schritt, hat sich seitdem bewährt und ist immer noch aktuell. Es bleibt spannend, die weitere Entwicklung von Java zu sehen.

Welchen Stellenwert besitzt die Java-Community für dich?

Haug: Die Java-Community mit ihrem regen Erfahrungsaustausch und der Bereitschaft, eine Vielzahl großartiger Open-Source-Projekte zu betreiben, hat meines Erachtens erst den großen Erfolg von Java ermöglicht.

Was hast du bei der Übernahme von Sun durch Oracle empfunden?

Haug: Ich habe das damals als große Chance gesehen – und sehe das heute noch so. Oracle hat neue Ideen in die Weiterentwicklung von Java eingebracht und steht auf einem soliden finanziellen Fundament, sodass ein langfristiger Entwicklungsprozess gewährleistet ist.

Wie sollte sich Java weiterentwickeln?

Haug: Mit einem gewichtigen Einbezug der Java-Community in die Weiterentwicklung von Java kann Oracle dafür sorgen, dass sich Java in allen wichtigen Bereichen weiterentwickeln wird und weiterhin eine breite Akzeptanz in der Entwicklergemeinde haben wird.

Wie sollte Oracle deiner Meinung nach mit Java umgehen?

Haug: Java ist eine herausragende Technologie mit einer unglaublich weiten Verbreitung in der IT-Welt. Dieses Gut sollte gehegt und gepflegt werden und damit erhalten bleiben.

Wie sollte sich die Community gegenüber Oracle verhalten?

Haug: Die Community sollte Oracle als den Hüter der Java-Technologie akzeptieren und sich weiterhin stark an der Weiterentwicklung beteiligen.

Andreas Haug
ah@jugm.de

Andreas Haug

Andreas Haug hat Diplom-Informatik an der Universität Ulm studiert. Sein beruflicher Werdegang führte ihn über verschiedene Softwareunternehmen der Finanzdienstleistungsbranche zu seinem derzeitigen Arbeitgeber mgm technology partners GmbH, wo er als Senior Software Architekt tätig ist. In seiner Freizeit beschäftigt er sich unter anderem mit Java-Technologien und als Autor.

Moving Java forward

Lucas Jellema, Amis; Paul Bakker, Open Source Amdatu PaaS Platform; Bert Ertman, Java User Group Leader for NLJUG, Netherlands

Anfang Oktober war die Woche der Wahrheit für Java: Die JavaOne 2011 fand zum zweiten Mal unter der Regie von Oracle in San Francisco statt. Seit Oracle das Sagen hat, findet die JavaOne zeitgleich mit der Oracle OpenWorld statt. Das hat Vorteile wie zum Beispiel das Konzert von Sting, stellt aber auch große Herausforderungen an den Charakter dieser Java-Konferenz. In diesem Jahr sollte sich zeigen, ob unter der Verantwortung von Oracle die Java-Plattform und die Java-Community wieder zunehmen, nachdem die Zahlen in den letzten Jahren bestenfalls stagniert haben. In diesem Artikel beschreiben Paul Bakker, Bert Ertman und Lucas Jellema – alle drei waren Sprecher auf dem JavaOne – wie sie die Konferenz erlebt haben und was für sie die wichtigsten Ergebnisse hinsichtlich der Zukunft von Java sind.

Gleich zu Beginn die größte Herausforderung: Eines der vorn anstehenden Probleme von Oracle nach der Übernahme von Sun ist der Umgang mit der Entwickler-Community. Keine andere Software-Entwicklungsplattform kennt eine derart bestimmende Community wie Java. Die Teilnahme an dieser Community ist etwas Besonderes. Für den Lieferanten hingegen macht das die Arbeit schwierig. Doch hat der Erfolg von Java viel damit zu tun, wie Sun in den vergangenen sechzehn Jahren mit der Community zusammengearbeitet hat. Ohne Community würde es heute kein Java geben. Nach der Übernahme von Sun will Oracle diesen Prozess steuern – ähnlich wie der Hersteller auch versucht, die Usergroups seiner anderen Produkte zu beeinflussen. Nicht zu vergessen: Die Java-Benutzer sind (potenzielle) Kunden für Oracle. Die letzten eineinhalb Jahre haben jedoch deutlich gezeigt, dass sich die Entwickler-Community nicht wie eine Usergroup steuern lässt. Hinzu kommt, dass Oracle, ein Unternehmen mit vielen Java-Entwicklern, den Ruf des „Abzockers" hat. Es sollte daher ein anderer Weg eingeschlagen werden, der schnell zum Erfolg führen sollte, sonst könnte Oracle die letzte Glaubwürdigkeit als Hüter von Java verlieren.

Nach dem Aufruhr im Vorfeld war klar, dass etwas geschehen musste. Oracle hat daher beschlossen, auf verschiedenen Gebieten aktiv zu werden und hat die Verantwortlichen der Java-Community (Java-User-Group-Leader und Java-Champions) zu strategischen Briefings eingeladen. Zudem hat Oracle monatliche Telefonkonferenzen organisiert, um nahe bei den Entwicklern zu sein und um zu erfahren, was die Community bewegt. Diese Art von Offenheit war neu für Oracle. Es war deutlich zu sehen, wie unbequem es für das Unternehmen war. Trotzdem wurde teilweise das verlorene Vertrauen wiedergewonnen. Die Anerkennung der Entwickler-Community ist ein ganz wichtige Punkt und ein Gewinn für die Java-Entwickler. Java ist ein technologisches Ökosystem, in dem neben Oracle auch andere Firmen und Personen eine Daseinsberechtigung haben. Oracle ist der Verwalter von Java, hat aber nicht das alleinige Sagen. Das Einbringen aller ist sehr wichtig für die Existenz dieser Plattform.

Das letzte Jahr stand im Zeichen der Klage gegen Android von Google und des Verlassens des JPC unter anderem durch

Abbildung 1: Adam Messinger, Vice President Development für Oracle Fusion Middleware (links), sowie Cameron Purdy, Oracle Vice President Development, auf der JavaOne 2011

Apache. Beide Themen wurden übrigens von den Offiziellen der JavaOne-Konferenz geschickt umgangen. Trotzdem sieht man, wie hinter den Kulissen die Rechtsabteilungen krampfhaft alles kontrollieren und nicht aus den Händen geben wollen.
Es gibt aber auch positive Entwicklungen: Mit „JCP.next" ist der JCP transparenter geworden. Es gibt wieder neue Versionen der Java-Plattform, die durch den JCP ratifiziert werden. Neue Gruppierungen nehmen am OpenJDK und beim JCP teil, das Ökosystem um Java braucht schließlich eine richtige Wiederbelebung. Noch ist nicht klar, wie groß der zuvor entstandene Schaden ist, aber das Wichtigste ist, der „Patient" lebt noch und kann weiterentwickelt werden. Java bewegt sich wieder vorwärts, das war ein guter Start in die JavaOne.

Abbildung 2: Treffpunkt der Community unter freiem Himmel in der Mason Street

Cloud

Der wichtigste Trend in diesem Jahr hieß eindeutig „Cloud". Obwohl das Thema bereits seit ein paar Jahren wichtig ist, sieht man jetzt, dass tatsächlich etwas entsteht, statt nur darüber zu reden. Ein intensiver Track mit Cloud-Themen, Best-Practice-Sessions von Early Adopters sowie von Cloud-Providern hat PaaS-Lösungen vorgestellt und in Java-EE-orientierten Sessions demonstriert, wie Cloud zu handhaben ist. Inzwischen ist klar, dass eine Cloud nicht nur für Benutzer mit extrem vielen Daten interessant ist. Auch die Skalierbarkeit ist ein wichtiges Thema. Mit PaaS verschwindet der Application-Server in der darunterliegenden Infrastruktur, was wichtige Voraussetzungen dafür schafft, wie Applikationen entwickelt und ausgerollt werden können. Es geht darum, das „ease-of-development" zu gewinnen. Das reduzierte, POJO-basierte Programmier-Modell mit vielen Java-APIs und Frameworks sowie die DevOps-Bewegung (kommt von Development und Operations) lieferten dazu einen wichtigen Beitrag.

Modularität

Nachdem es in den letzten Jahren den Anschein hatte, OSGi-Technologie sei auf der JavaOne verboten, gab es in diesem Jahr eine gute Sichtbarkeit. Neben reinen OSGi-Themen standen mehrere Sessions über Modularität im Allgemeinen auf dem Plan. Jeder scheint inzwischen damit einverstanden zu sein, dass Modularität gewährleistet sein muss, insbesondere wenn Applikationen in einer Cloud laufen sollen. Für die genaue Modularitäts-Lösung ist aber noch einiges zu klären. Alle Augen sind dabei auf das Jigsaw-Projekt gerichtet, das Teil von Java 8 ist.

Jigsaw startete vor ein paar Jahren, um den JDK selbst modular zu machen. Der JDK ist durch Hinzufügen von immer mehr Features und APIs in den letzten zehn Jahren stark gewachsen und hat mittlerweile einen bedeutenden Umfang.

Bei manchen APIs stellt sich allerdings die Frage, wie sinnvoll diese noch sind, weil sie in modernen Applikationen fast nicht mehr verwendet werden. Bestes Beispiel dafür ist die Unterstützung von Corba. Andererseits ist es unmöglich, diese APIs einfach zu entfernen, weil dann die Rückwärtskompatibilität fehlschlägt, und die Applikationen, die abhängig davon sind, werden dann zur ewigen Anwendung einer der älteren Java-Versionen verurteilt.

Die einzige Lösung ist, das JDK modular zu gestalten, damit der Anwender nur die JDK-Teile herunterladen muss, die zum Starten einer Applikation wichtig sind. Auf JDK-Ebene ist OSGi aber keine Lösung, weil OSGi eine Ebene darüber steht. Die jetzigen Pläne für Jigsaw gehen aber viel weiter als bis zum Aufteilen von JDK. Die Unterstützung der Modularität wird auch für Java-Entwickler nützlich sein, um eigene Applikationen modular zu machen.

Jigsaw löst das Problem mit einem „bundle" mit der OSGi-Ebene. Wo OSGi dafür ein externes Framework und ein extra Metadaten-Format benötigt, bekommt Jigsaw Unterstützung aus der Virtual-Machine-Sprache. Zurzeit wird heftig diskutiert, ob die Art, wie Jigsaw das angeht, die richtige ist. Momentan sieht es danach aus, dass Jigsaw und OSGi zwei separate Lösungen werden, die Zusammenarbeit dieser beiden jedoch in Zukunft möglich sein wird. Die Meinungen darüber, was dies künftig für OSGi bedeutet, laufen auseinander. In jedem Fall bietet Jigsaw keine Lösung auf der Services-Ebene in OSGi. Das ist sicherlich der OSGi-Bereich, der die meisten Vorteile für das richtige modulare Gestalten von Applikationen bietet. Eine Kombination aus Modulen mit Basis-Sprachunterstützung und darauf aufsetzenden OSGi-Services könnte sicherlich eine sehr interessante Sache sein. Das letzte Wort dazu ist sicher noch nicht gesprochen.

Java SE 8

Aus verschiedenen Gründen hat sich das Release von Java SE 7 erheblich verzögert. Nachdem seit der Version 6 mehr als fünf Jahre vergangen sind, wurde Java SE 7 mit nur wenigen Erneuerungen herausgebracht, während die großen Änderungen

EINIGE FAKTEN UND ZAHLEN

Wie im letzten Jahr fand die JavaOne zusammen mit der Oracle OpenWorld statt. Beide Events sind an San Francisco nicht ganz spurlos vorbeigegangen – es wurden insgesamt rund 46.000 Besucher gezählt, die meisten (ca. 40.000) davon auf der OpenWorld. Im weiten Umkreis der Stadt war die ganze Woche kein Hotelbett mehr zu bekommen. Die zahlreichen Besucher kamen nicht nur zu beiden Konferenzen, sie brachten auch ziemlich viel Taschengeld mit. Nach Angaben der Stadt wurden in dieser Woche ca. 100 Millionen Dollar extra ausgegeben – ein bedeutender Anteil davon im lokalen Apple Store. Die OpenWorld konzentrierte sich auf das Moscone Convention Center und ein paar große Hotels in direkter Umgebung. Die JavaOne fand einige Blocks weiter nordwestlich statt. „The Zone" nennt sich das Dreieck aus drei großen Hotels, wobei das Hilton im Mittelpunkt steht. Die Mason Street zwischen den Hotels war gesperrt. Dort war für die JavaOne-Besucher eine Lounge-Area aufgebaut, in der soziale Aktivitäten stattfinden. Oracle gibt die konkrete Anzahl der Besucher offiziell nicht bekannt, Insider sprechen von etwa 6.000. Damit hätte sich die Besucherzahl im Vergleich zum Vorjahr verdoppelt. Es gab rund 400 Vorträge, logistisch war einiges verbessert. Gleichartige Sessions fanden oft im gleichen Raum statt, in jedem Fall aber im gleichen Hotel. Wer an allen Aspekten von Java interessiert war, hatte einige Kilometer zurückzulegen.

erst mit Java SE 8 kommen sollen. Java SE 7 steht mittlerweile seit Juli 2011 zur Verfügung. Während der JavaOne ist auch erwähnt worden, dass die offizielle Version für Mac OS X für Ende 2011 geplant ist und direkt darauf Anfang 2012 ein General-Availability-Release. Übrigens ist der Technology Compatibility Kit (TCK), mit dem bestimmt wird, ob ein JVM tatsächlich SE-7-zertifiziert wird, noch nicht fertig. Oracle nennt als Grund dafür das Fehlen von Anwälten und kündigt den TCK noch für das vierte Quartal 2011 an.

Weil Java SE 8 große Erneuerungen bringen soll, die zeitaufwändig sind, und weil große Firmen bei Oracle und beim OpenJDK-Projekt verkündet haben, dass eine Periode von ein bis zwei Jahren zwischen zwei wichtigen Java-Releases zu kurz ist, wird Java SE 8 bis zum Summer 2013 erwartet. Die wichtigsten Themen werden dabei die Modularität (Projekt Jigsaw) sowie das Projekt Lambda (closures und bulk parallel operation in Java collections sowie filter/map/reduce) sein.

Durch die Verschiebung des Release-Datums können in Java SE 8 auch Funktionalitäten integriert werden wie die neue Date-Time-API (JSR 310, auch bekannt als JodaTime), kleinere Verbesserungen in der Sprache, Security Updates, erneutes Networking, Type Annotations, eine neuere JavaScript-Engine (Projekt Nashorn) sowie Support für moderne Device-Funktionen wie Multi-Touch-User-Interface, Kompass-Integration-Tool sowie Kamera-Lokation-Bestimmung. Ein wichtiges Ziel bei Java SE 8 ist für Oracle die verbesserte Interaktion zwischen verschiedenen JVM-Sprachen. Die neue JavaScript-Engine soll außerdem ein gutes Beispiel sein für die Interaktion, die dann auch mit anderen JVM-Sprachen wie Groovy oder Scala nutzbar ist. Es sieht ferner danach aus, dass in Java SE 8 das Projekt Avatar (siehe unten) auch Funktionalitäten für HTML5 und WebSockets bringen wird. Mit dem OpenJDK-Projekt stehen Teile davon kurzfristig in Early-Access-Downloads zum Selbst-Entdecken zur Verfügung. Dies ist wiederum ein Zeichen der Community mit dem Ziel, Dinge zu ändern. Die gesamte Entwicklung von Java ist damit transparent und öffentlich – unfassbar für die übliche Produkt-Entwicklung innerhalb von Oracle.

Oracle hofft, kurzfristig nach Java SE 8 die JavaFX-3.0-API als Teil der SE-Spezifikation zum kurzfristigen Nachfolger der jetzigen Swing-API machen zu können. Das Hinzufügen von JavaFX (siehe unten) über die Erneuerungen auf diesem Gebiet wird ziemlich kontrovers diskutiert. Bisher wurde Java FX vollständig außerhalb von JCP und OpenJDK von Oracle selber entwickelt und ist zurzeit kein Bestandteil des Java-Standards. Wenn Java FX kurzfristig Swing als Standard für die UI-Entwickelung auf der Java-Plattform ersetzen soll, ist es wichtig, dass es Teil der Java-SE-Plattform wird. Dadurch muss es allerdings zuerst den JCP passieren, was nicht nur eine Formalität ist. Oracle ist zwar Verwalter von Java, hat aber nicht das alleinige Sagen innerhalb des JCP. Hier wird sich zeigen, wie gut der JCP funktioniert.

Es gibt auch schon Pläne für Java 9. Ein interessantes Thema dabei ist eine sich selbst tunende Virtual Machine, um die Abhängigkeiten von einigen (manchmal unklaren) Command-Line-Parametern zu verringern. Andere Themen, die in Gerüchten kursieren, sind – neben selbstverständlich weitergehender Modernisierung und Verbesserung von Java als Programmiersprache – die Integration verschiedener Sprachen auf dem JVM und eine bessere Integration mit Native (O/S) Libraries und Bestimmungen. Ganz der NoSQL-Bewegung folgend sind Bestimmungen geplant für „Big Data", große In-Memory-Daten-Sammlungen, die die jetzige Java-Speichergrenze von zwei GB für Arrays erhöhen soll. Auch soll Java 9 nach jetzigen Plänen eine neue Meta-Objekt-Beschreibung bieten, um einfachen Objekt-Austausch zwischen Sprachen zu ermöglichen. Mehr Unterstützung für Advanced Concurrency auf Multi-Core-Plattformen ist ebenfalls denkbar. Hinzu kommt „reification", ein Tool, um den Information-Type von Objekten unter „runtime" zur Verfügung zu stellen und die von vielen gewünschte Korrektur um die ursprüngliche Implementation von Generics in Java SE 5 zu ermöglichen.

Multi-Tenancy steht auch auf der Liste für Java 9, um in Cloud-Umgebungen gleichzeitig mehrere Applikationen innerhalb der gleichen JVM ablaufen zu lassen. Damit verwandt ist die Unterstützung für das Ressourcen-Management bei Cloud-Applikationen. Gleichzeitig ist dieses Thema auch in der nächsten Java-EE-Version geplant.

„Continuations" ist eine Programmiersprache basierend auf Lambda-Expressionen, die einen fast asynchronen Request von lokalen Methoden unterstützt, wobei

eine Request-Methode keinen Wert wiedergibt, sondern vom „Requestor" ein Objekt erhält, in dem die Ergebnisse platziert werden können. In Java 9 wird erwogen, Continuations zum Teil von Java zu machen.

Gleichzeitig ist geplant, Java SE und ME wieder zusammenzuführen – mit einem eingeschränkten (core) Footprint, der auf allen Plattformen zur Verfügung stehen wird. Ein weiteres Ziel der näheren Zukunft liegt darin, dass wieder überall (von Card bis zu EE und Cloud) die gleiche Java-Sprache angewendet werden kann.

Die Kooperation mit dem OpenJDK-Projekt sieht auch sehr gut aus. Neben Oracle, RedHat, IBM, SAP und Apple sind auch Azul und Twitter vor Kurzem hinzugekommen. Der Community-Process scheint sich auch hier wieder zu bewegen.

Java EE 7

Java EE 6 ist mittlerweile fast zwei Jahre alt. Inzwischen ist Java EE durch sieben Container unterstützt und damit häufig als Mainstream-Technologie in der Praxis anzutreffen. Die meisten Container sind sehr leicht und starten in wenigen Sekunden, was ein sehr großer Entwicklungsschritt gegenüber den traditionellen Applikations-Servern bedeutet. Das ist wiederum ein wichtiger Schritt in Richtung „Cloud". In der Cloud wird eine Applikation oft nach Nutzen abgerechnet. Das unnötige Einsetzen von Speicher kostet Geld und ein kleiner Memory-Footprint ist damit sehr wichtig geworden. Daneben ist eine schnelle Startzeit essenziell für das richtige Hoch- und Runterskalieren eines Clusters in der Cloud. Obwohl Java EE 6 damit jetzt erst für die meisten Entwickler interessant wird, ist die Entwicklung von Java EE 7 bereits in vollem Gang. JSRs und Expert-Groups werden gebildet und neue APIs veröffentlicht. Manche Spezifikationen wie JAX-RS 2.0 sind bereits größtenteils klar. Das Hinzufügen von APIs ist in einigen Container-Implementierungen bereits integriert, an anderer Stelle sind noch Änderungen erforderlich, wenn beispielsweise JAX-RS-2.0 die Löcher stopfen wird, über die Entwickler in der Praxis gerade stolpern. Für andere Spezifikationen wird noch über neue Ideen nachgedacht. Ein Beispiel dafür ist die Multi-Tenancy-Unterstützung in JPA. Hier ist noch Input aus der Community erforderlich, um die richtige Form zu finden. Der Ruf dazu erfolgte auch während der JavaOne. Die meisten JSRs sind noch offen und das Feedback ist einfach zu geben, genauso wie das Beteiligen an einer Mailingliste. Ein anderes wichtiges Statement über Java EE 7 besagt, dass es nicht nur um die Cloud geht. Trotz der Cloud-Unterstützung werden bei Java EE 7 sicherlich auch andere Bereiche der Plattform verbessert. Man denke beispielsweise an CDI 1.1, JMS 2.0 und EJB 3.2. Auch eine Java-API für das Manipulieren von JSON steht auf dem Programm. Wenn es um die Cloud geht, sind folgende Themen wichtig für Java EE 7:

- Multi-Tenancy
- Cache/Data-Grid APIs
- NoSQL
- Deployment

JavaFX

JavaFX war auf JavaOne prominent positioniert. Dabei wurde vor allem deutlich, dass JavaFX nicht länger eine Spielzeug-Technologie ist, sondern strategisch wichtig für Oracle. So war es auch keine Überraschung, dass ein paar wichtige Ankündigungen dazu gemacht wurden:

- JavaFX wird im Laufe des nächsten Jahres vollkommen Open Source, ähnlich wie OpenJDK
- Es kommt eine Spezifizierung (JSR) für JavaFX
- Die Mac-Version von JavaFX erscheint noch in diesem Jahr (die Beta-Version steht bereits zur Verfügung)
- Die Linux-Version kommt Anfang 2012

In den Keynotes sind ein paar beeindruckende Demos von JavaFX-Applikationen gezeigt worden. Vor allem die 3D-Demos mit importierten Modellen zeigten, wie stark diese Technik auf dem grafischen Gebiet ist. Es waren auch Demos zu sehen, bei denen JavaFX-Applikationen auf einem Tablet PC abliefen. Die Unterstützung des iOS steht aber noch nicht auf der Roadmap, die Community wird noch zu deren Wichtigkeit befragt. Wie schon letztes Jahr verkündet, ist JavaFX jetzt eine richtige Java-Technologie. JavaFX-Script gibt es nicht

REST

In vielen Gesprächen auf der JavaOne war „REST(ful)" aus unterschiedlichen Gründen ein wichtiges Thema. REST ist einfacher als SOAP/WS-WebServices und deshalb beliebt. Daneben kann REST in Verbindung mit JSON als Datenformat einfach auf mobilen Devices und in JavaScript-Web-Clients eingesetzt werden. Die JavaOne zeigt einen weiteren Grund für REST – die Cloud. Wenn Java-Applikationen vermehrt in der Cloud-Infrastruktur eingesetzt werden, ist auch die Administration von JVMs in der Cloud mehr und mehr gefragt. Für das Daten-Management dienen neben den bekannten Application-Servern und webbasierten Konsolen auch programmatische Mittel, basierend auf MBeans und JMX, EJBs oder auf „op-proprietary"-Protokollen wie t3 bei WebLogic. Diese Mechanismen funktionieren nicht über HTTP in der Cloud. RESTful APIs sind dann eine naheliegende Alternative.*

REST an sich ist nicht neu und die JAX-RS-1.1-API ist Teil von Java EE 6. In JSR 339 wird JAX-RS 2.0 als Untermenge von JEE 7 entwickelt. Die betreffende Expert-Group umfasst neben Oracle unter anderem auch Ebay, RedHat und Talend. Mit JAX-RS 2.0 geht das Erwachsenwerden von REST weiter. Eigene Kern-Elemente in der JAX-RS-2.0-Spezifikation sind bessere Client-Libraries für RESTful Services, standardisierte Filter und Handler für Standard Interception und die Bearbeitung von ein- und ausgehenden Berichten, Validation von Parametern und Ressourcen mithilfe von Bean Validation, asynchrone Interaktion auf Basis der asynchronen Bearbeitung von Servlet 3.0 und die Unterstützung für Links auf andere RESTful-Ressourcen.

mehr und die JavaFX-Applikation wird jetzt einfach in Java geschrieben. Dazu stehen APIs zur Verfügung: eine traditionelle API, die ähnlich aussieht wie Swing, ein auf dem Builder-Pattern basierende API und eine XML-basierte Variante, um Markups zu machen. Damit können Entwickler mit unterschiedlichem Background gut mit APIs arbeiten. Weil keine separate Sprache mehr existiert, können auch alle IDEs wieder direkt für das Entwickeln von JavaFX-Applikationen angewendet werden.

Swing ist für tot erklärt. Die politisch korrekte Aussage dazu ist, dass momentan nur noch Bugs behoben und kaum neue Entwicklungen gemacht werden. Zum Glück funktioniert die Integration zwischen Swing und JavaFX und klare Migrationswege sind im Kommen. JavaFX ist damit nicht mehr nur als RIA-Konkurrent für Silverlight und Flex positioniert, sondern muss als Standard-Java-GUI-Technologie der Zukunft gesehen werden. Die Art, wie GUI angeboten wird (Desktop oder Web), ist bestimmt durch den darunterliegenden „renderkit". Für das Web müssen wir weiterhin Java Web Start und Applets nutzen. Für die Zukunft steht Rendering zu HTML 5 auf dem Programm.

Projekt Avatar: HTML 5 und WebSockets

In der Strategie-Keynote kündigte Oracle das Projekt Avatar an. Grund dafür ist die Entwicklung von HTML 5. Avatar ist auf den ersten Blick ein unklares Projekt für eine sehr eindeutige Entwicklung. HTML 5 wird zunehmend als die Ablösung von Flash für Rich-Visualisierungen in Browsern und auf mobilen Geräten gesehen. Sowohl zwischen den Web-Servern und Web-Clients oder mobilen Clients als auch zwischen den Servern selbst – HTML 5 wird oft mit dem Aufstieg des WebSockets-Protokolls als leichtes, bidirektionales Kommunikationsprotokoll, das echten Push ermöglicht, assoziiert. Mit dem Projekt Avatar möchte Oracle sicherstellen, dass für alle Teile der Java-Plattform (SE, EE, ME und JavaFX) HTML 5 und WebSockets unterstützt werden und dass dieser Support konsistent ist.

Das Projekt Avatar ist „eine komplette Lösung für dynamische Rich-Client-Anwendungen" und nennt nachdrücklich Java EE (Cloud) als Server und für HTML-5-Browser-Anwendungen, hybride Anwendungen von HTML 5 und Java in einem Browser oder auf mobilen Geräten und (Desktop-) Anwendungen als Client mit Kommunikation basierend auf JSON über WebSockets. Die bidirektionale Kommunikation sollte innerhalb Avatar genutzt werden, um zu Event-basierten Anwendungen zu kommen, unter anderem mit Benachrichtigungen bei Datenänderungen. Avatar umfasst auch eine standardisierte Methode von Offline-Möglichkeiten für Web-Anwendungen. Da es hier speziell um „Slide-Ware" ging und keine konkreten Beispiele gezeigt wurden, ist noch abzuwarten, wie Avatar irgendwann Einfluss auf die Entwicklung der Java-Plattform

JMS 2.0

Letztes Jahr gab es auf der JavaOne 2010 ein Community-Treffen, auf dem abends um zehn Uhr in einer Birds-of-a-Feather-Session (BoF) etwa 40 Personen über die Zukunft von JMS philosophierten. Zu dieser Zeit war Java EE 7 noch nicht erschienen und es gab immer noch keine Klarheit über eine JMS-Version 2.0. In dieser BoF-Session wurden Ideen ausgetauscht, etwa darüber, wie JMS verbessert werden könnte und wo die Schwachstellen sind. Jetzt, ein Jahr später, ist Java EE 7 in vollem Gange und es gibt in der Tat eine JMS-2.0-Spezifikation (JSR 343), erstellt von einer Expertengruppe, in der unter anderem Tibco, RedHat, VMware, Caucho, IBM und Oracle vertreten waren. Während der JavaOne 2011 wurde wieder in einer neuen BoF-Session diskutiert.

Ziel dieses JSR ist die Bereinigung der alten JMS-1.1-Schnittstellen aus dem Jahr 2003, die Modernisierung wie bei Java EE 5 und 6 (mit Annotations, Unchecked Exceptions und Injection), die Unterstützung von Funktionen, die in vielen Messaging-Produkten zur Verfügung stehen – wie Minimum Delivery Delay, Bestätigungsnachrichten von Message Delivery, Delivery Count, Batch Delivery –, und das Lösen einiger MDB-Unvollkommenheiten. Eine Folge der vorgeschlagenen Änderungen würde unter anderem bedeuten, dass dieser Code:

```
public void sendMessage (String payload) {
try {
  Connection conn = null;
  con = cf.createConnection();
  Session sess =conn.createSession(false,Session.AUTO_ACKNOWLEDGE);
  MessageProducer producer = sess.createProducer(dest);
  TextMessage textMessage = sess.createTextMessage(payload);
   messageProducer.send(textMessage);
} catch (JMSException e) {
  // do something
} finally {
connection.close();
}
}
```

umgeschrieben werden kann in:

```
@Inject @JMSConnection(lookup="jms/connFactory") @JMSDestination(lookup="jms/
inboundQueue") MessageProducer producer;
public void sendMessageNew(String payload){
  producer.send(payload);
}
```

haben wird. Der Gedanke dahinter ist aber sicherlich wichtig, um Java als Plattform zusammen mit dem Web-Programmiermodell zu entwickeln.

Oracle und das Vermächtnis von Sun

Oracle als Betreuer von Java scheint (in der zweiten Chance) nicht zu enttäuschen. Die Beziehungen innerhalb der Java-Community stabilisieren sich und die Zahlen der Teilnehmer in JCP und OpenJDK erhöhen sich. Die Zusammenarbeit zwischen kommerziellen Konkurrenten wie IBM, Apple, Oracle, SAP und RedHat / JBoss entwickelt sich in Bezug auf die Java-Plattform fruchtbar – abgesehen von der etwas unglücklichen Situation mit Google, der fehlenden Partei bei dieser Konferenz.

Oracle hat das Vermögen der Sun Microsystems jetzt eingehend untersucht. Einige Teile des Nachlasses wurden liebevoll in die Arme geschlossen (wie JavaFX, NetBeans und GlassFish), andere auf das Abstellgleis gesetzt (wie JCAPS, OpenESB, Project Kenai und jMaki). Die Stimmen aus der ehemaligen Sun-Welt sind weitgehend positiv. Nach Jahren von begrenzten Möglichkeiten aufgrund der finanziellen Lage von Sun ist nun für ausgewählte Projekte deutlich mehr Raum für Investitionen, Expansion und Wachstum. Die Stagnation, zu der es seit 2008 in allen Bereichen von Java kam (Plattform und Community), scheint gebrochen. Davon profitieren wird fast jeder.

Man betrachte zum Beispiel die Sun Hotspot JVM: Oracle hatte durch die BEA-Übernahme mit JRockit bereits eine JVM im Besitz. Diese war im Handel erhältlich für anspruchsvolle Umgebungen, in denen umfangreiche Management- und Feinabstimmungseinrichtungen erforderlich waren. Auch die Garbage Collection sollte vorhersehbar und besser angepasst sein. Oracle hat entschieden JRockit und Hotspot in einer JVM zu verschmelzen – mit den Vorteilen und Stärken von beiden. Diese Konvergenz findet im OpenJDK statt und wird in ein paar Schritten durchgeführt bis zu oder bis nach JDK 8. Zuerst kommen das JCMD Command-Line-Tool, JMX-Agent und MBeans, Java Discovery Protocol (JDP) und Flight Recorder (schon vor JDK 7) an die Reihe. Eine kommerzielle JRockit-Ausgabe mit Enterprise-Unterstützung bleibt erhalten, aber auch ohne diese Unterstützung ist JRockit jetzt schon unter der gleichen Lizenz verfügbar und damit nutzbar als freie Hotspot-JVM.

GlassFish ist ein weiteres Produkt, das sich mit einem kommerziellen Produkt von Oracle überlappt, mit dem von BEA übernommenen WebLogic Server. GlassFish ist die Referenz-Implementierung der Java EE und damit ein Eckpfeiler der Spezifikation. Sun hat GlassFish sowohl als Open-Source-Applikations-Server als auch mit einer Enterprise-Support-Lizenz angeboten. Die Community war neugierig auf Oracles Pläne.

Auch im Hinblick auf GlassFish gilt, dass Oracle nicht gern zwei vollständige Java EE Application Server unterstützen möchte. Auf der anderen Seite will Oracle mit dem WebLogic Server tatsächlich Geld verdienen, während man für die Java-EE-Spezifikation als Referenz-Implementierung GlassFish verpflichtet ist. Die Lösung ist hier, dass die GlassFish- und die WebLogic-Server-Teams eng zusammenarbeiten und gegenseitig Funktionen und Bibliotheken austauschen und gemeinsam nutzen. Somit ist die JSF-Bibliothek in WebLogic und in GlassFish identisch und wird ebenso rund um den Metro-Web-Service-Stack-Code geteilt. Mit der Zeit wird sich herausstellen, ob WebLogic eine Erweiterung von GlassFish wird – als eine kommerziell angebotene Bereicherung mit erweiterten Funktionen wie Coherence WebCache, Clustering und Verwaltung, basierend auf GlassFish mit Java EE.

Neben einer Referenz-Implementierung ist GlassFish auch ein Schaufenster für neue Ideen in Bezug auf die Änderungen, die notwendig sind, um Java EE 7 Cloud-ready zu machen. Dies bietet Input für die JCP-Expertengruppen, die sich damit beschäftigen werden. GlassFish ist damit ein vielseitiges Produkt und hat den Status „Spielplatz für ein Java-EE-Produkt" seit Langem verlassen. Dies bedeutet in jedem Fall, dass GlassFish auch unter der Obhut von Oracle – oder vielleicht sogar stärker als je zuvor – als vollständiger, unterstützter und bezüglich Features führender Java-EE-Container eingesetzt werden kann. Die Verbreitung von GlassFish könnte damit stärker steigen.

„Wenn wir uns heute entscheiden müssten, hätten wir nie damit anfangen", so der Kommentar einiger Java-Verantwortlicher bei Oracle. „Aber nun, da wir es in den Schoß gelegt bekommen haben, wäre es wirklich schade, sich nicht in vollem Umfang an dem Einstieg zu beteiligen." Es geht hier um JavaFX – die von Sun während der JavaOne 2008 präsentierte Technologie für das Benutzer-Interface von morgen. Trotz all des Trubels rundum und des Interesses an JavaFX funktioniert die Technik nicht wirklich berauschend. Kurz vor der Übernahme von Sun bezeichnete Larry Ellison in seiner Keynote auf der JavaOne JavaFX als eines der interessantesten Schmuckstücke in der Sun-Kollektion. In der Tat hat Oracle großartig dort weitergemacht, wo Sun aufgehört hat. JavaFX-Script wurde fallengelassen und durch Java-APIs ersetzt. Dies ist ein erster Schritt in Richtung der Aufnahme von JavaFX in SE, wodurch JavaFX definitiv als Nachfolger und Ersatz von Swing und AWT begrüßt werden kann. Nicht nur auf den Desktop, sondern auch innerhalb von Applets, mit HTML-5-Rendering in üblichen, Browser-basierten Seiten und vielleicht auch wieder auf mobilen Geräten sollte die JavaFX-UI Java-Technologie-of-Choice werden. Ob es mit JavaFX funktioniert, ist sicherlich nicht nur von Oracle abhängig – aber auch dieses Erbstück ist ausgezeichnet.

Die Übernahme von BEA hat Oracle den Workshop-Plug-in für Eclipse gebracht, von Sun kam sogar eine komplette IDE hinzu, nämlich NetBeans. Wer gedacht hatte, dass Oracle bei drei IDEs (JDeveloper, Eclipse und NetBeans) sehr bald rationalisieren würde, irrte sich gewaltig. Alle drei IDEs sind weiterhin unterstützt. NetBeans ist und bleibt die Plattform für die Popularisierung der Java-Plattform in all ihren Facetten und wird deswegen als Erstes mit der Unterstützung für die neuesten Versionen von Java SE, Java EE und JavaFX ausgestattet. NetBeans wird von Oracle auch als IDE für andere JVM-Sprachen wie JRuby, Groovy und Scala positioniert. Es ist aber Aufgabe der Community, die Nicht-Java-Unterstützung in der NetBeans-Plattform zu definieren.

NetBeans 7.1 wurde während der JavaOne als Beta-Version freigegeben. Diese bietet unter anderem die Unterstützung für JavaFX 2.0. Weitere neue Features sind CSS3-Unterstützung, Tools für visuelles

Debugging von Swing- und JavaFX-Benutzeroberflächen, die in die IDE integrierte Unterstützung für Git sowie neue PHP-Debugging-Funktionen. Neben NetBeans entwickelt Oracle auch Oracle Enterprise Pack for Eclipse (OEPE) weiter – einschließlich der Integration mit dem WebLogic Server (da liegen dann auch die kommerziellen Interessen) – und es gibt eine immer mehr zunehmende Unterstützung für ADF-Entwicklung (vielleicht in Erwartung einer bevorstehenden Community-Edition). Die JDeveloper-IDE ist die dritte in der Reihe. Sie bleibt die strategische Entwicklungsumgebung für die Oracle-Fusion-Middleware-Produkte – von ADF über WebCenter-Suite bis SOA, BPM Studio und Data Integrator.

Fazit

Nach Jahren mit sehr minimaler Weiterentwicklung oder fast Stillstand, was im Vergleich zu anderen Technologien oder dynamischen Sprachen wie .NET oder iOS/Objective C eigentlich einen Rückschritt bedeutet, ist die Java-Plattform im allen Bereichen wieder in Bewegung. Das zeigt sich im Release von SE 7 (selbst auf Mac OS X), den Plänen für SE 8, den sehr konkreten Entwicklungen und Ambitionen mit EE 7 und dem ernsthaften Weiterentwickeln von JavaFX. Oracle scheint die Verantwortung, die die Rolle als Verwalter von Java mit sich bringt, nun ernst zu nehmen – auch aus klarem Eigeninteresse. Es gab tatsächlich eine Rationalisierung, in deren Rahmen manche Projekte gestrichen wurden, aber gleichzeitig hat Oracle die Investitionen in andere Projekte im Vergleich zu Sun stark anwachsen lassen. Die Versprechungen aus dem Jahre 2010 bezüglich der Weiterführung von Initiativen und konkreten Auslieferungen sind zum größten Teil eingehalten worden. Nach einigem Erkunden und hier und da einem Knistern ist eine breit angelegte und wachsende Community mit etlichen großen Anbietern – vor allem IBM, aber auch Twitter, Azul, SAP und Red Hat / Jboss, VMWare / SpringSource – zusammen mit Oracle (selbst wenn die Kontroversen rund um Google unglücklich sind) entstanden.

Die Zahlen rund um Java bleiben beeindruckend: Die Anzahl der Geräte, auf denen Java in der einen oder anderen Form aktiv ist, sowie die globale Reihe von Entwicklern und Anbietern ist enorm. Wer darauf besteht, dass Java zum Scheitern verurteilt oder sogar schon gestorben ist, ist ein Nachplapperer von suggestiven Blogs oder schaut nicht auf die wirklichen Fakten.

Oracle sollte dann auch in den kommenden Jahren alles daran setzen, dafür zu sorgen, dass die Community einbezogen wird. Konkrete Entwicklungen schaffen das Vertrauen, dass Java erneut wieder dabei ist, eine moderne Sprache und Plattform zu sein. Um modern weiterzuarbeiten, schicke Sachen zu machen, attraktive Benutzeroberflächen zu entwickeln, um die neueste Technologie zu erschließen und um von den modernen Kenntnissen in Programmiersprachen und Software-Entwicklung zu profitieren, kann man einfach mit der Java-Plattform weitermachen.

Lucas Jellema
lucas.jellema@amis.nl

Paul Bakker
paul.bakker@luminis.eu

Bert Ertman
bert.ertman@luminis.eu

Ins Deutsche übertragen von Michel Keemers und Daniel Dibbets

Lucas Jellema ist Oracle ACE Director für Fusion Middleware. Er arbeitet als CTO, Consultant, Trainer and Autor bei Amis in den Niederlanden.

Paul Bakker ist Senior Developer bei Luminis Technologies in den Niederlanden. Er arbeitet an der Open Source Amdatu PaaS Plattform.

Bert Ertman ist Partner von Luminis in den Niederlanden. Er leitet die niederländische Java User Group Leader (NLJUG) und ist Sun/Oracle Java Champion.

Weitere Neuvorstellungen im Rahmen der JavaOne und Openworld

Neben JavaFX 2.0 hat Oracle in San Francisco eine Reihe weiterer Neuigkeiten vorgestellt, die auch für Entwickler interessant sind:

- Die Oracle Exalytics Business Intelligence Machine ist das weltweit erste In-Memory-Hard- und Software-System für schnellere Analysen als je zuvor. Die Lösung bietet visuelle Analysen in Echtzeit und enthält neue Typen von Analyse-Anwendungen.
- In Vorschau auf das geplante Release im November 2011 präsentiert Oracle Solaris 11. Das führende Betriebssystem für Unternehmen bietet höchste Verfügbarkeit, Sicherheit und Leistungsfähigkeit sowohl auf SPARC als auch auf x86 Systemen. Entwickelt für Clouds, erlaubt es den sicheren und außergewöhnlich schnellen Einsatz in großen Cloud-Umgebungen.
- Oracle Application Development Framework (ADF) Mobile ist eine Erweiterung des Oracle Application Development Framework und wird die Entwicklung von Java-basierten mobilen Anwendungen weiter vereinfachen. Als Teil von Oracle Fusion Middleware wird es mit Oracle ADF Mobile möglich, native und Web-Funktionen in einer mobilen Anwendung zu vereinen. Basierend auf einem Next-Generation Framework erleichtern die neuen Funktionen die Entwicklung mobiler Anwendungen weiter.
- Mit Verfügbarkeit von Oracle Exastack Optimized erweitert Oracle sein Exastack-Programm. Es erlaubt Independent Software Vendors (ISVs) und anderen Mitgliedern des Oracle PartnerNetwork, ihre Anwendungen auf Oracle Exadata Database Machine und Oracle Exalogic Cloud in punkto Geschwindigkeit und Zuverlässigkeit zu optimieren.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN
0700 11 36 24 39

ODER BESTELLEN SIE ONLINE
go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:
Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.